UTRECHT UNIVERSITY

MASTER'S THESIS

Pareto Local Search For Multi-Objective Land-Use Allocation

Author:
G. DE JONGE

Supervisor:
Dr. Ir. D. THIERENS
Second Examiner:
Dr. F. van der HILST

A Thesis Submitted In Fulfillment Of The Requirements For The Degree Of Master Of Science

of the

Master's Programme Computing Science
Department of Information and Computing Sciences

July 15, 2020

UTRECHT UNIVERSITY

Abstract

Pareto Local Search For
Multi-Objective Land-Use Allocation

by G. DE JONGE

Multi-objective land-use allocation (MOLA) is a multi-combinatorial optimization problem, in which land-use types are allocated to units of land to optimize multiple objectives while satisfying imposed constraints. MOLA techniques are among others used by researchers and land planners to assist in planning sustainable land-use. As land-use change is currently an important human driver of environmental degradation, the demand for MOLA techniques has increased. Main challenges in this field are the scalability of the spatial optimisation process to enable the optimisation for larger areas and multiple objectives and to develop realistic solutions. This research examines the potential of a local search based algorithm Pareto Local Search (PLS) for MOLA to address these issues. In this research, a modular (iterated) PLS algorithm for MOLA, named (I)PLS-MOLA, was designed and implemented. Together with this algorithm, three search operators for MOLA, two repair operators and a new data structure for storing MOLA solutions were set up. The proposed algorithm, operators and data structure were applied to a case study that concerns land-use optimization in Brazil, which could provide insight in how Brazil could plan its future land-use to meet the future food, feed and biofuel demands in a sustainable manner. Eventually, the performance of the PLS-MOLA and IPLS-MOLA algorithm were compared to the performance of NSGA-II, which is currently the most used optimization technique for MOLA. The results show that PLS-MOLA and IPLS-MOLA prove to be more scalable than NSGA-II, with a more efficient and better converging optimization process. Especially with larger MOLA problems, the solution quality of PLS is significantly higher than the solution quality of NSGA-II. Combined with the fact that PLS can also search more efficiently for solutions that can emerge from the current situation, assuring more realistic solutions, the algorithm proves to be a promising technique for future MOLA research.

Preface

Before you lies the bachelor thesis "Pareto Local Search For Multi-Objective Land-Use Allocation". It has been written to fulfill the graduation requirements of the master's degree in Computing Science at Utrecht University. The thesis has been written in the period between November 2019 and July 2020.

The project has evolved from the demand of F. van der Hilst from the Energy and Resources group for an efficient technique to optimize land-use allocation in Brazil. The group has executed research in land-use change and sustainability for several years, and recently became interested in applying multi-objective optimization techniques. As evolutionary algorithms are currently the most applied technique in this field, Dr. Ir. D. Thierens was involved. Eventually, this exploratory research project was set up to examine the current techniques available and sort out which ones are most effective. During this research, the PLS algorithm came into the picture as an interesting and promising technique to optimize MOLA problems. Eventually, I am happy to say that an algorithm has been designed, implemented and compared that shows promising results for not only the Brazil case, but MOLA problems in general.

I would like to thank Dr. Ir. D. Thierens for his guidance throughout the project. Additionally, I would like to thank Dr. F. van der Hilst for her expertise regarding land-use optimization in Brazil.

I hope you enjoy your reading.

Guus de Jonge

Utrecht, July 15, 2020

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AC	Acceptance Criterion / Angle Crossover
ACD	Average Crowding Distance
ARI	Average Recurrence Interval
ASC	Action Script Communication
BC	Block Crossover
BCRM	Biased Cell Repair Mutation
BCX	Boundary Cell Crossover
BPI	Best Pareto Improvement
BRCM	Biased Random Cell Mutation
BRCSM	Biased Random Cell Swap Mutation
BUC	Block Uniform Crossover
CD	Crowding Distance
CGPGA	Coarse-Grained Parallel Genetic Algorithm
COP	Combinatorial Optimization Problem
DM	Decision Maker
EA	Evolutionary Algorithm
FPI	First Pareto Improvement
GA	Genetic Algorithm
GHG	Greenhouse Gas
GP	Goal Programming
GPLS	Genetic Pareto Local Search
HBC	Horizontal Band Crossover
HC	Horizontal Crossover
IDE	Integrated Development Environment
IO	Input - Output
IPLS	Iterated Pareto Local Search
KBRM	K-Tournament Boundary Cell Repair Mutation
KCRM	K-Tournament Cell Repair Mutation
KRBM	K-Tournament Random Boundary Cell Mutation
KI	Knowledge Improved
KRPM	K-Tournament Random Patch Mutation
KRCM	K-Tournament Random Cell Mutation
LP	Linear Programming
LR	Local Repair
LS	Local Search
LUM	Land-Use Management
MAGNET	Modular Applied General Equilibrium Tool
MCDA	Multi-Criteria Decision Analysis
MD	Metadata
MSIS	Mutation for Steering Infeasible Solution
MOCO	Multi-Objective Combinatorial Optimization
MOGA	Multi-Objective Genetic Algorithm
MOLA	Multi-Objective Land-Use Allocation
MOLU	Multi-Objective Optimization of Land-Use
MOPSO	Multi-Objective Pareto Swarm Optimization

MPLS	Multi-Restart Pareto Local Search
MR	Macroregion
NE	Neighborhood Exploration
NIMBY	Not In My Back Yard
NPI	Neutral Pareto Improvement
NSGA-II	Non-Dominated Sorting Genetic Algorithm II
OHI	Optimistic Hypervolume Improvement
PAES	Pareto Archive Evolutionary Strategies
PBIO	Problem-Based Initialization Operator
PC	Personal Computer
PLS	Pareto Local Search
POC	Proof Of Concept
PSA	Pareto Simulated Annealing
RAM	Random Access Memory
RBCM	Random Boundary Cell Mutation
RBCRM	Random Boundary Cell Repair Mutation
RBIM	Random Block Initialization Mutation
RBM	Random Block Mutation
RCLM	Random Cluster Mutation
RCLRM	Random Cluster Removal Mutation
RCM	Random Cell Mutation
RCRM	Random Cell Repair Mutation
RCSM	Random Cell Swap Mutation
RDCO	Regional District of Central Okanagan
RPM	Random Patch Mutation
RPRM	Random Patch Repair Mutation
RPSM	Random Patch Swap Mutation
RQ	Research Question
SA	Simulated Annealing
SDSS	Spatial Decision Support Systems
SOGA	Single-Objective Genetic Algorithms
SOP	Single-Objective Optimization Problem
SPM	Single-Point Mutation
SPX	Single-Point Crossover
TDX	Two-Dimensional Crossover
TEL	Total Edge Length
TS	Tabu Search
UC	Uniform Crossover
UML	Unified Modeling Language
VBC	Vertical Band Crossover
VC	Vertical Crossover
XBC	Boundary Cell Crossover
XTD	Two-Dimensional Crossover

Phase I

Exploration and Preparation

Chapter 1

Introduction

1.1 Introduction

Multi-objective land-use allocation (MOLA) is an optimization problem in which land-use types are allocated to land units (cells), subjecting to a set of objectives and constraints. (Song and Chen, 2018). MOLA can be found in many domains, such as urban, land-use and local authority planning (Huang, Liu, Li, et al., 2013). In each domain, the so-called planner is challenged to come up with an optimal type allocation for the involved land units. These land units can be variables in a vector or cells in a grid, managed by a geographic information system (GIS). MOLA is a combinatorial optimization problem (COP), of which the solution space can be significantly large and has been described as an NP-hard problem. As often many stakeholders are involved, land-use allocation can be a complicated process in which formulating the problem and ending up with a final plan that satisfies all involved parties can be a difficult task (Schwaab, Deb, Goodman, et al., 2018). Several often-used objectives are the maximization of the compactness, the suitability and the economic profitability. (Masoomi, Mesgari, and Hamrah, 2013). As these objectives are often contradictory, only rarely one solution is found that is optimal to each objective. Therefore, the 'solution' is often a set of solutions known as the Pareto optimal set (Pareto front). In order to assist decision makers in a MOLA case, several techniques have been developed to come up with scenarios that approximate this Pareto front. These tools are often called Spatial Decision Support Systems (SDSS), and are build upon techniques from Geographical Information Science (GIS) and Multi-Criteria Decision Analysis (MCDA).

Over the previous decades, multiple MCDA techniques have been proposed for SDSS's to approximate the requested Pareto front (Huang, Liu, Li, et al., 2013). Earlier attempts tried to rephrase MOLA into a single-objective optimization problem (SOP). This was done by for instance addressing weights to the objectives and summing them up. By repeatedly altering the weights and optimizing the resulting problem with single-objective optimization techniques, the Pareto front could be approximated. At first, linear programming (LP) was applied to find solutions for MOLA (Aerts, Eisinger, Heuvelink, et al., 2003). However, there were two distinct limitations to this approach: one is that it can not completely take into account spatial objectives whose values vary non-linearly with cells' attribute values; the other is that it is helpless in handling regions with more than 50 x 50 cells because of the numerous variables and constraints that emerge (Li and Parrott, 2016). Other techniques implement single-objective search heuristics together with a weighted sum or goal programming approach. This includes the implementation of heuristics such as simulated annealing (SA), tabu search (TS) and single-objective

genetic algorithms (SOGAs) (Aerts, Herwijnen, and Stewart, 2003) (Sharma, 2005) (Cao and Ye, 2013). SA en TS are probabilistic techniques that apply local search to approximate the global optimum, whereas SOGAs tend to evolve a pool of solutions to an optimum using an evolution based approach. Although these approaches scale significantly better than LP, they still suffer from several drawbacks. These drawbacks are mainly the result of applying an a priori approach to a multi-objective problem. First of all, the resulting solutions can be unevenly distributed and secondly, some sections of the Pareto front (especially the concave part) might be approximated badly. Also, the single-objective problem (SOP) resulting from the a priori approach, may need to be solved repeatedly to eventually approximate the complete front. Although these drawbacks can partly be overcome by using strategies that steer the convergence to less explored areas, the resulting scalability is still too weak for interactive SDSS systems (Cao and Ye, 2013).

In order to overcome these issues, multi-objective search heuristics have been applied to MOLA which are able to approximate the complete Pareto front in one execution. Various multi-objective optimization algorithms have been proposed, such as Pareto Archive Evolutionary Strategies (PAES), Multi-Objective Particle Swarm Optimization (MOPSO) and Multi-Objective Genetic Algorithms (MOGAs) (Huang, Liu, Li, et al., 2013). If applied well, these algorithms are able to approximate the Pareto front of multi-objective optimization problems (MOPs) with complicated factors, such as a large number of candidate solutions, non-linearity and complicated objectives. Among these attempts, evolutionary algorithms (EAs) have been applied to MOLA most often. One of the first applications of MOGAs to land-use planning was done in (Matthews, Craw, Elder, et al., 2000). Afterwards, many others followed. Efforts to boost the performance of genetic algorithms for MOLA have resulted in a variety of challenges, such as the optimization of several non-standard parameters. (Schwaab, Deb, Goodman, et al., 2018). Examples of these, are the population size, crossover and mutation probabilities and stopping criteria. However, most challenging is to adapt a GA to the spatial nature of MOLA, asking for crossover and mutation operators that are able to efficiently exchange genes. Also, setting up an initial population that allows for an efficient convergence to the Pareto front can be a difficult task. The MOGA that has been applied to MOLA most often, is the Non-Dominated Sorting Genetic Algorithm (NSGA-II). This algorithm implements a domination based selection mechanism that has proven to be efficient in many multi-objective combinatorial optimization problems (Datta, Deb, Fonseca, et al., 2007).

In this research, the use of a multi-objective local search (MOLS) heuristic for MOLA will be examined. The possibilities of MOLS for MOLA have been research earlier, for instance by using PSA in (Duh and Brown, 2007). However, the amount of research to properly adopt these search heuristics to MOLA in an efficient manner, is still relatively low. This research will further elaborate on the possibilities of MOLS by exploring in more depth how a MOLS algorithm can be applied to MOLA efficiently. The most straightforward approach of multi-objective local search, is the Pareto Local Search (PLS). PLS has proven to be an efficient method for multi-objective combinatorial optimization (MOCO) (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015). It was

first proposed and experimentally tested in (Paquete, Chiarandini, and Stützle, 2004) for the multi-objective traveling salesman problem. PLS extends the often-used single-objective hill-climbing algorithm, by turning it into a multi-objective optimization algorithm (Cabrera-Guerrero, Mason, Raith, et al., 2018). In PLS, the set of the most optimal solutions found are stored in an archive. Repeatedly, the neighborhood of one of these solutions is explored, after which the archive is updated. In order to globally explore the search space, the algorithm can be extended to an Iterated PLS (IPLS) algorithm. This algorithm iteratively calls PLS, using a perturbed solution from a previous run as the input. By perturbing these solutions, the algorithm aims to escape local optima. The PLS algorithm has a completely different approach than the previously discussed techniques, which could bring several interesting advantages. The PLS and IPLS algorithm move through the solution space in a more controllable manner (smaller steps), which can decrease the number of constraint violations. Besides, the PLS algorithm is able to more efficiently explore solutions similar to the current situations, which is sometimes desired by stakeholders. More details on the challenges that current algorithms are facing, and the advantages that PLS could bring will be discussed in the Problem Description of Chapter 1.2. As PLS has never been applied to MOLA before, this research will first examine how PLS and IPLS can be applied to MOLA efficiently. In order to do so, different strategies and operators being used in the algorithm will have to be optimized. These components are for instance responsible for the exploration of a neighborhood or reparation of an invalid solution. All PLS components that are being examined to explore the potential of PLS for MOLA are stated in the Research Questions of Chapter 1.3. Each question is accompanied with a small explanation of why it is relevant to exploring the possibilities of PLS in the MOLA domain. Eventually, the algorithm will be compared with the currently most used optimization technique for MOLA, namely NSGA-II. The results will be analyzed in order to bring a clearer view on the actual effectiveness of PLS for MOLA, pointing out whether PLS could be a promising technique for current applications and future MOLA research.

1.2 Problem Description

The aim of optimization techniques in the domain of MOLA, is to enable the realization of functional SDSS's that can add value to the decision making process. In order to do so, MOLA techniques are needed that provide both a high scalability and approximation quality (Li and Parrott, 2016). The approximation quality concerns how close the eventual Pareto front resembles the optimal Pareto front. In order to add value to the decision-making process, a minimum approximation quality will need to be assured. The scalability concerns the running time of the algorithm for a given problem size to obtain a certain approximation quality. Generally, it is undesirable to let the running time scale exponentially with the problem size. Currently, most solutions are especially experiencing difficulties with respect to the scalability. First of all, the linear programming (LP) algorithms, which already require extremely long running times with normal sized MOLA problems (Aerts, Eisinger, Heuvelink, et al., 2003). Secondly, the single-objective algorithms combined with a priori approaches, such a SA, TS and several GAs, which need multiple runs to eventually deliver the Pareto front (Duh and Brown,

2007) (Sharma, 2005) (Cao, Huang, Wang, et al., 2012). Finally, the multi-objective GAs for MOLA, among which the NSGA-II algorithm (Deb, Pratap, Agarwal, et al., 2002) is the most popular. These bring the challenge of designing qualitative operators that allows for an efficient convergence towards the Pareto front. Unfortunately, current NSGA-II approaches still result in relatively high running times. Besides enabling functional SDSS's, current techniques also bring the disadvantage of always trying to explore the complete search space. In case a stakeholders is interested in how the current situation could or should emerge, only solutions close to the current situation will need to be examined. This can be enforced with objectives and constraints, but as the performance generally drops as the number of objectives increases, this is not desirable.

As discussed before, the PLS and IPLS algorithm are deemed interesting for solving MOLA cases due to their completely different optimization approach. Namely, by only taking small steps in the search space, the algorithms are able to alter solutions in a much more controlled manner. In this way, except for when perturbing a solution, the number (and size) of constraint violations can be kept relatively low. For MOGAs such as NSGA-II, it is more difficult to move through the search space without without causing violations. This would require crossover operators that are able to effectively exchange genes while respecting the constraints, which are difficult to set up for large sized problems. The disadvantage of more constraint violations is that it requires more 'repair' operations to fix these invalid solutions, which can be costly. As so, IPLS is expected to be more scalable than NSGA-II and optimize MOLA problems more efficiently. Besides the improved scalability, PLS is also expected to be more effective in generating solutions comparable to the current situation. As it is a local search algorithm, the PLS algorithm first explores the solution space close to the initial solution and converges to one or more local optima nearby. This obviates the need of extra objectives or constraints to enforce similarities with the current solution, increasing the efficiency. Due to these two expected advantages, PLS could be an interesting option for MOLA optimization. The potential of the PLS and IPLS algorithm will be further explored in this research, to bring insight in whether and to what extent PLS could indeed bring answers to the challenges currently being faced within the MOLA domain.

1.3 Research Questions

The overarching goal of this exploratory research project is to explore the potential of PLS in the MOLA domain. The 'potential' of PLS is indicated by the efficiency of PLS compared to currently used MOLA techniques. The 'efficiency' is hereby measured in terms of the hypervolume obtained after a certain running time. In case the efficiency of an algorithm is higher than the state-of-the-art algorithms, it will be considered as 'efficient'. In this research, the overarching research question will be whether the PLS algorithm is an efficient algorithm for solving MOLA problems.

Is PLS an efficient algorithm for MOLA?

In order to answer this overarching question, four research questions have been formulated. The four research questions are explained in more detail

below. The first question focuses on how PLS can be applied to MOLA most efficiently. To do so, six subquestions have been formulated to examine the efficiency of different strategies and operators. The second question examines the scalability in terms of the number of objectives to view the impact of adding more objectives on the efficiency. Next, the third question concerns how PLS can most efficiently be transformed to a global IPLS algorithm. Finally, the fourth question asks for a comparison between the efficiency of IPLS and NSGA-II, which is currently the most applied MOLA technique. Together, these research questions are able to provide an answer on the overarching question of whether PLS is an efficient algorithm for MOLA.

1. **How can PLS most efficiently be applied to MOLA?**
   The first step in examining whether PLS is an efficient algorithm, is to optimize PLS for MOLA problems. The overall outline of the PLS algorithm will be based upon the setup of Paquete, Chiarandini, and Stützle, 2004. However, the design of its data structures, strategies, operators and criteria being used, will be optimized for MOLA problems specifically.

   (a) **What is an efficient data structure for storing MOLA solutions?**
   The most straightforward manner to store a MOLA solution is by naively storing the land-use type of each cell of the map. As many new solutions are being created (and copied) in the optimization process of PLS, this could result in large memory demands. This subquestion examines what data structure(s) could be used to store solutions more efficiently and reduce those memory costs.

   (b) **What is the influence of the allocation range size on the running time?**
   The most common MOLA constraint, states to how many cells each land-use type should be allocated (Chapter 3.2). Often, this is done using a lower and upper bound. The larger the size of this range, the less violations (and so 'reparations') are expected to occur, which could decrease the time to perform an equal number of iterations. However, this also decreases the control of the user. This subquestion examines the relationship between range size and running time.

   (c) **What is an efficient strategy for selecting the next solution to be explored?**
   In the standard PLS algorithm, the solution of which the neighborhood will be explored next, is chosen at random among the unexplored ones. This subquestion will explore whether the usage of an alternative selection strategy that selects the least crowded solution (Chapter 5.3.1, 6.4 and 6.5), could improve upon the efficiency of PLS for MOLA.

   (d) **What is an efficient strategy for exploring the neighborhood of a solution?**
   A variety of operators and procedures can be used to explore a solution its neighborhood. Several of these will be discussed in Chapter 5.3.3 and 6.6. This subquestion will address which operators and procedures result in an efficient search (either separately or combined) when implemented in PLS for MOLA.

   (e) **What is an efficient strategy for repairing non-feasible solutions?**
   Different reparation operators and procedures can be used to repair solutions violating the problem constraints. Several will be discussed in

Chapter 6.9 and 6.10. This subquestion will address which operators and procedures will result in an efficient reparation of solutions.

(f) **What are efficient archive acceptance criteria for non-dominated solutions?**
The standard PLS algorithm adds a solution to archive in case it is not dominated by any solution currently in the archive. However, different criteria have been introduced to manage the quality and size of the archive as will be discussed in Chapter 5.3.2 and 6.11. This subquestion will examine the effects of different criteria on the efficiency of PLS for MOLA.

2. **How does the efficiency of PLS for MOLA scale with the number of objectives?**
The PLS algorithm is able to solve multiple objectives at once. However, the more objectives are being optimized, the lower the optimization efficiency per objective (individually) generally becomes. It might therefore be preferable to, for instance, combine multiple objectives into one. This question will tend to examine the impact of the number of objectives on the optimization efficiency.

3. **How can PLS most efficiently be processed into a global IPLS algorithm for MOLA?**
Different variants of PLS have been proposed to improve upon the overall performance. One of these variants is Iterated PLS, and will be discussed in more detail in Chapter 5.2.2. This question will comprise how the PLS algorithm can efficiently be processed into an IPLS algorithm for MOLA by looking at aspects such as the convergence speed of PLS and the perturbation size.

4. **How does IPLS compare to NSGA-II in terms of efficiency when applied to MOLA?**
As the NSGA-II algorithm is currently the most applied and researched technique for optimizing MOLA problems, IPLS will be compared to NSGA-II. The NSGA-II algorithm will be implemented according to the previous research discussed in Chapter 4.3.2. The algorithm will hereby be optimized by incorporating the neighborhood exploration and reparation operators that have proven to be most efficient in Research Question 1D and 1E.

1.4 Contribution & Relevance

MOLA is currently a hot topic, as the search towards an algorithm that enables a high-quality interactive SDSS is in full swing (Schwaab, Deb, Goodman, et al., 2018). Since the current techniques are not able to live up to the demands, existing algorithms for MOLA are continuously being improved while new ones are being introduced (Li and Parrott, 2016). The currently most popular technique, namely NSGA-II, especially performs poorly in terms of scalability. This is partly caused by the fact that designing efficient crossover operators that respect the constraints tends to be a difficult task. As discussed before in Chapter 1.2, designing efficient search operators that take relatively small steps in the search space can be designed more easily. Therefore, local search heuristics become an interesting option for MOLA, possibly bringing an outcome for the current scalability issues. Currently, the amount of research towards techniques incorporating multi-objective local search heuristics for MOLA is relatively low compared to other directions. By examining MOLS for MOLA, more insight can be gathered in whether and to what extent this local search based approach could contribute to the development

of scalable MOLA techniques. Furthermore, the research goes beyond the general PLS framework and examines its separate components as well. This includes the design of an efficient data structure, neighborhood exploration operators, selection procedure, archive acceptance criteria and exploration strategy. Since these components can be exchanged between different algorithms, these findings will be interesting for other MOLA techniques as well. Furthermore, different variants of PLS will be tested and eventually compared with NSGA-II. Since this is the most popular algorithm currently used for MOLA, the performance of PLS will be put in perspective to the current state-of-the-art techniques being used.

Finally, the results might contribute to the research being performed at the Energy and Resources group of Utrecht University. This research group is examining land-use changes due to the increasing global demands in biofuel and their effect on the environment. An efficient multi-objective optimization technique could assist this group in getting more insight in how these changes could (ideally) be handled more sustainably. One of the projects of this institute focuses on the situation in Brazil, which will be be translated to a MOLA problem and serve as a case study for the PLS algorithm. As so, the case study being used will be a realistic now-a-days MOLA problem. More details of the MOLA problem in Brazil can be found in Chapter 8.2. The outcome of this research will eventually aid in the exploration and selection of an optimization approach suitable for this research project. All software generated for this research will optionally be transferred to this institute.

1.5 Research Outline

The research will be executed in the period between November 2019 and July 2020. At first, a literature research will be executed towards multi-objective land-use allocation in general. In Chapter 3.1, 3.2 and 3.3, the problem context, definition and five of the most common objectives will be handled. This will be followed by an outline of several a priori and a posteriori approaches for MOLA in Chapter 3.4 and different representations of MOLA solutions in Chapter 3.5. Finally, ten example MOLA cases are summarized in Chapter 3.6.1, of which the two most common ones are described in more detail. After handling multi-objective land-use allocation in general, a literature research will be executed to previous and current optimization techniques being used for MOLA. First, in Chapter 4.1, the application of linear programming is discussed. Next, in Chapter 4.2, several local search heuristics will be handled. This will be followed by an extensive summary of genetic algorithms for MOLA in Chapter 4.3, including both single- and multi-objective GAs together with their initialization procedures, selection procedures, crossover operators and mutation operators. Finally, a third literature research will be executed towards Pareto Local Search in general. At first, the definition and several variants will be discussed in Chapter 5.1 and 5.2. Next, the most important components/procedures are discussed in Chapter 5.3, followed by its anytime behavior in Chapter 5.4.

After the literature research has been completed, a Pareto Local Search algorithm will be designed for MOLA. At first, the algorithm setup will be defined in Chapter 6.2, which includes the solution representation and data structures being used. Next, the PLS and IPLS algorithms will be defined

in Chapter 6.3, of which the most important components will be handled in the following chapters. In Chapter 6.4 and 6.5, the selection procedure and two selection operators will be discussed. In Chapter 6.6, two neighborhood exploration strategies will be discussed based upon the search procedure of Chapter 6.7, and three search operators will be defined in Chapter 6.8. Next, the reparation procedure and two reparation operators will be discussed in Chapter 6.9 and 6.10, after which several archive acceptance criteria will be outlined in Chapter 6.11. Finally, incremental solution updating and validating procedures will be discussed in Chapter 6.12 and 6.13 to eventually conclude with the perturbation procedure for IPLS in Chapter 6.14. The next step in the research, is to implement the PLS algorithm for MOLA that has been designed. The general setup and frameworks used to build the algorithm will be described in Chapter 7.1 and Chapter 7.2. Next, the UML diagrams showing how the algorithm can be implemented in an objective-oriented programming language will be shown and discussed in Chapter 7.3. This includes the implementation of both PLS and IPLS, followed by an NSGA-II variant for MOLA. Finally, the parameters of the algorithms will be discussed in Chapter 7.4 and eventually concurrency (parallelism) within these algorithms will be handled in Chapter 7.5

After the algorithms have been implemented, several experiments will be set up and executed to answer the research questions of this research. The general setup of these experiments will be discussed in Chapter 8. Next, a problem case will be defined together with F. van der Hilst of the Energy and Resources group of Utrecht University. The problem case will be set up and discussed in Chapter 8.2. This includes the problem context, definition and subcases. Next up, several test cases will be proposed to answer the research questions in Chapter 8.3. Each test describes what algorithm runs should be executed, what data should be measured, and how this can aid in answered one or more research questions. Eventually, the materials that will be used to run the experiments will be summarized in Chapter 8.4. In the final phase, all experiments will be executed and the required data will be gathered. The results of the experiments will then be summarized and discussed in Chapter 9.2, 9.3, 9.4 and 9.5. Throughout this chapter, the research questions will be answered. In Chapter 10 a conclusion will be given regarding all research questions and the overall exploratory goal of the research. Finally, based upon the discussed results and conclusions of the research, recommendations for future research will be outlined in Chapter 11. These recommendations will eventually be passed on the Energy and Resources group of Utrecht University, which will continue to work in this research direction if deemed interesting.

Chapter 2

Preliminaries

Most real world problems cope with multiple, often conflicting, objectives simultaneously (Abraham and Jain, 2005). Consider for example, the optimization of an electric car that has to be both fast and have a long driving range. Improving on one objective could degrade another objective, resulting in multiple 'best' solutions. The interest in optimizing such multi-objective problems has increased significantly over the last three decades, as it can be used to deliver better (automated) decision making in all kinds of domains, ranging from finance to healthcare to artificial intelligence (Abraham and Jain, 2005). However, multi-objective problem optimization brings multiple challenges compared to single-objective optimization. When there are two or more conflicting objectives, one will not be able to find one unique optimal solution anymore. There will now be a set of multiple solutions that are all of equal quality (Abraham and Jain, 2005). A multi-objective optimization problem can be defined as follows (Zitzler, Deb, Thiele, et al., 2001):

$$\begin{aligned} &\text{Minimize}\{f_1(x), f_2(x), ..., f_k(x)\} \\ &\text{subject to } x \in S \end{aligned} \tag{2.1}$$

With $k \geq 2$ objective functions $f_i : \mathbb{R}^n \to \mathbb{R}$ that we are all minimizing at the same time. The decision vector $x = (x_1, x_2, ..., x_n)^T$ belongs to the feasible region $S \subset \mathbb{R}^n$. This feasible region is formed by constraint functions. The elements of the feasible region $Z \subset \mathbb{R}^k$ are called the objective vectors. Note that maximizing f_i is equivalent to minimizing $-f_i$ (Zitzler, Deb, Thiele, et al., 2001). When all objectives and constraints are linear functions, the problem is a nonlinear multi-objective optimization problem. In case one or more of these functions are nonlinear, it is considered to be a nonlinear multi-objective optimization problem.

When the objective functions contradict each other, it will not be possible to find one single solution that optimizes all objectives simultaneously. In multi-objective optimization, a solution is said to be dominated by another solution, if the other solution can improve on at least one of its objective values without degrading any of the other objective values. If solution s dominates s', this is notated as $s \prec s'$. A solutions is non-dominated, if it is dominated by no other solution; this is called Pareto optimality.

Definition 2.0.1. Decision vector $x \in S$ is Pareto optimal if there is no other $x^* \in S$ such that $f_i(x^*) \leq f_i(x)$ for all $i = 1, ..., k$ and $f_j(x*) < f_j(x)$ for at least one index j (Zitzler, Deb, Thiele, et al., 2001).

The set of all Pareto optimal solutions is called the Pareto optimal set. The solutions of this set are all equally acceptable solutions of the multi-objective optimization problem. However, in real world situations, it is often desired

to end up with one single solution. Selecting one solution out of the Pareto optimal set calls for a decision maker (DM) (Zitzler, Deb, Thiele, et al., 2001). Therefore, finding the final decision usually means a collaboration between the decision maker and the analyst. The analyst is a person or program that is accountable for the information, for instance an approximation of the Pareto optimal set, which is used by the DM to decide upon the final solution.

Chapter 3

Multi-Objective Land-Use Allocation

3.1 Problem Context

One of the multi-objective optimization problems that has gained an increased interest lately, is the multi-objective land-use allocation problem (MOLA). The MOLA problem is defined as the process of allocating land-use types to units of land to meet a set of objectives of the stakeholders (Datta, Deb, Fonseca, et al., 2007). Currently, land-use changes are an important human driver of environmental degradation on local, regional, and global scales (Li and Parrott, 2016). Therefore, the demand for effective tools to assist future land planners has increased. In order to realize these tools, effective heuristics for solving and/or approximating the multi-objective land-use allocation problem are needed. Over the years, multiple approaches to this problem have been introduced and evaluated, which will later on be discussed in Chapter 4.

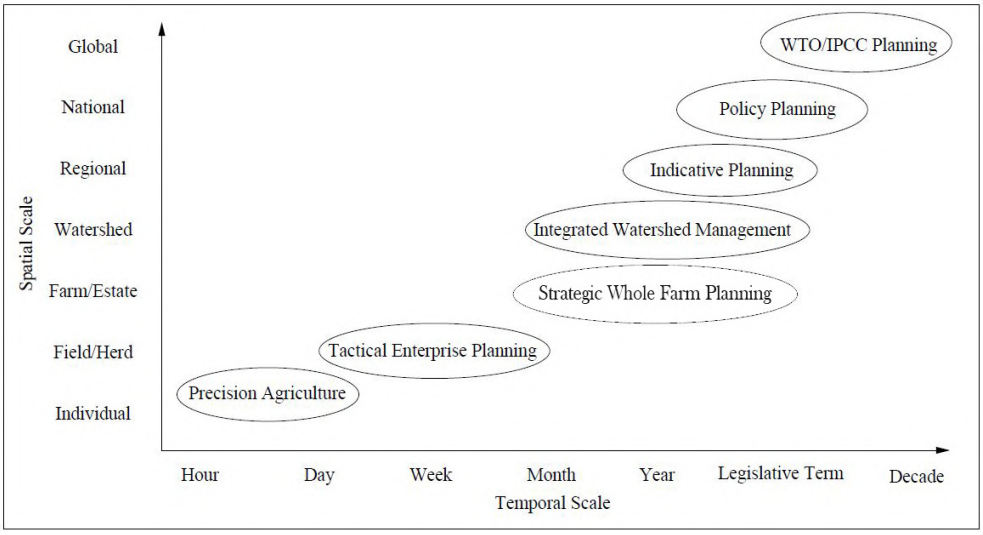


FIGURE 3.1: MOLA Domains On Spatio-Temporal Land-Use Planning Scales (Matthews, Craw, Elder, et al., 2000)

According to (Matthews, Craw, Elder, et al., 2000), several land-use planning domains can be distinguished, characterised by their spatial and temporal scales. The spatial scales range from the individual plants and animals to global ecologies, whereas the temporal scales range from hour(s) to decade(s). Several domains including their position on these scales are shown in Figure 3.1. In most researches, the focus is on domains that are either regional or national on the spatial scale, and in between a year and a decade on the temporal scale. Some of these cases will be discussed in Chapter 3.6.

3.2 Problem Definition

The basic MOLA problem is formulated as follows. We have a rectangular area, which we will divide into a grid (N rows and M columns). Next, we have K land-use types, denoted by $k = 1, ...K$ that can be allocated to one cell. We therefore define a binary variable x_{ijk}, that is equal to 1 in case k is given to cell (i, j), and 0 otherwise. Parameter T_k represents the number of cells in the grid that need to be given a type k. Finally, B_{ijk} equals the parameter of an objective, of which its value differs per location. The MOLA model can now be formulated as follows (Cao, Huang, Wang, et al., 2012) (Cao, Batty, Huang, et al., 2011):

Minimize:

$$\sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} B_{ijk} x_{ijk} \tag{3.1}$$

Subject to:

$$\sum_{k=1}^{K} x_{ijk} = 1 \quad \forall i = 1, ..., N, j = 1, ..., M \tag{3.2}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M} x_{ijk} = T_k \quad \forall k = 1, ..., K \tag{3.3}$$

where:

$$x_{ijk} \in \{0, 1\} \quad \forall i = 1, ..., N, j = 1, ..., M, k = 1, ..., K \tag{3.4}$$

Equation (3.1) can occur multiple times, with different values of B_{ijk} set for each objective. In case only one objective is used, the problem becomes a single-objective land-use allocation problem. In equation (3.2), it is specified that a land-use type needs to be given to every land unit. Next in equation (3.3), the number of land units to which each type should be allocated are set. (Aerts and Heuvelink, 2002). Alternatively, the exact numbers of T_k can be replaced with percentages P_k that states the percentage of cells that should be of type k. Equation (3.3) would than become as follows:

$$\sum_{i=1}^{N} \sum_{j=1}^{M} x_{ijk} = NMP_k \quad \forall k = 1, ..., K \tag{3.5}$$

Often, a different land-use restriction is used instead of Equation (3.3) or (3.5), that allows the number of cells to which a land type is assigned to be in a certain range (Cao, Huang, Wang, et al., 2012). The number of land units to which type k can be allocated is then allowed to be in between a lower bound L_k and upper bound U_k. This results in the following constraint.

$$L_k \leq S_k \leq U_k \tag{3.6}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{M} x_{ijk} = S_k \quad \forall k = 1, ..., K \tag{3.7}$$

$$\sum_{k=1}^{K} S_k = NM \tag{3.8}$$

Hereby, S_k is the total number of cells to which type k is allocated. Equation (3.8) forces that the total number of cells to which a land type is allocated, is equals the number of land units in the grid.

Finally, the problem does not necessarily have to be notated using a grid of N times M cells. Other notations are possible as well, such as a numbering system that assigns a unique number $i = 1, ...N$ to each cell. Then, a cell can be noted as x_i (Cao and Ye, 2013). In Chapter 3.5, more different problem representations will be discussed.

3.3 Problem Objectives

In reality, objectives are concretized by the urban planners and other related stakeholders. However, in this chapter several of the most frequently returning objective functions will be outlined.

3.3.1 Minimizing Development Costs

The basic and most used objective of the land-use allocation program, is the minimization of development costs (Aerts, Eisinger, Heuvelink, et al., 2003). Hereby, the development costs C_{ijk} are simply the costs for assignment land-use type k to cell (i, j). The objective then becomes as follows:

Minimize:

$$\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{M} C_{ijk}x_{ijk} \tag{3.9}$$

In case of a single-objective land-use allocation problem, this is often the (only) objective. Another naming for development costs is conversion costs, as in most cases you are actually conversing an already existing land-use type to a new land-use type (Cao, Batty, Huang, et al., 2011).

3.3.2 Maximizing Suitability

A more extended version of the development costs objective, is the 'suitability' objective. Now, the 'costs' of assigning land-use is specified more extensively according to multiple decisive factors. The suitability of a single cell is then calculated using the weights given to each of these factors. Factors considered by (Masoomi, Mesgari, and Hamrah, 2013) are the area (A), the accessibility (Ac),the number of edges per cell (Ed), the slope (S), the ownership type (P), the sound pollution (Sp), the air pollution (Ap) the resistance to change (R) and the difference between edge sizes (De). More factors can

be added or removed up to the stakeholder's preferences. The suitability S_{ik} of land-use type k for cell i can now be defined as follows:

$$S_{ik} = w_1 A_{ik} + w_2 Ac_{ik} + w_3 Ed_{ik} + w_4 S_{ik} + w_5 P_{ik} + w_6 Sp_{ik} + w_7 Ap_{ik} + w_8 R_{ik} + w_9 DE_{ik} \tag{3.10}$$

where w_1 to w_9 are the weights that are assigned to each suitability factor. These are defined as follows:

$$\sum_{i=1}^{9} w_i = 1 \tag{3.11}$$

As the units and concepts of the factors are different, these are translated to comparable scales and normalized to the range $[0, 1]$. Finally, the objective can be formulated as follow:

Minimize:

$$\frac{1}{n} \sum_{i=1}^{n} S_{ik} \tag{3.12}$$

The weights given to each factors, can be determined by urban planners and involved stakeholders. As can be noted, it can take some effort to do so for each land-use type for each cell. Therefore, this objective is not very convenient to use. More often, its factors are replaced by separate objective functions, of which the values are calculated in a more dynamic manner. For example, the accessibility objective, which is discussed later in Chapter 3.3.4.

3.3.3 Maximizing Compactness

Generally, it is not desired (as it is impractical) to have small patches of land-use being scattered all over the map. As so, large and continuous sections of equal land-use are often favored. (Aerts and Heuvelink, 2002). Therefore, in addition to the development costs objective, it is common to add an objective that encourages compactness of land-use. The eight-neighbour objective for maximizing compactness is defined as follows (Li and Parrott, 2016):

Maximize:

$$\sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(x_{ijk} \sum_{m=i-1}^{i+1} \sum_{n=j-1}^{j+1} \frac{Neig_{mn}}{8} \right) \tag{3.13}$$

where

$$Neig_{mn} = \begin{cases} 1, & \text{if } x_{ijk} = x_{mnk} \\ 0, & \text{otherwise} \end{cases} \tag{3.14}$$

If the land-use type of a neighbour cell (m, n) of cell (i, j) is the same, the the value of $Neig_{mn}$ will be 1, and 0 otherwise (Li and Parrott, 2016). In Equation 3.13, diagonal cells are also counted as neighbouring cells, making this

called the eight-neighbour method. A more simpler variant of the compactness objective is the four-neighbour method, that only counts the four directly neighbouring cells. An example of the four-neighbour objective can be found in (Aerts, Eisinger, Heuvelink, et al., 2003). In this research, we will work according to the four-neighbour system (which is more simple), unless stated differently.

There are multiple other objective functions possible to stimulate compactness. For instance, a minimization of the number of clusters formed per land-use (Aerts, Herwijnen, and Stewart, 2003). Minimizing this number, would promote the formation of larger and more continuous clusters. Another possibility, is a minimization of the perimeters (number of edges) of each cluster. This value is often divided by the square root of its area, resulting in the following objective function:

Minimize:

$$\sum_{c=1}^{C} \frac{H_c}{\sqrt{s_c}} \tag{3.15}$$

with H_c is the perimeter and s_c is the area size of cluster $c = 1, ...C$. In this research, we will not go into further detail on this approach. For more details, view (Aerts, Herwijnen, and Stewart, 2003).

Additionally, constraints can be added to stimulate the convergence towards a compact solution. For example, it is generally undesirable to have single cells as clusters in the final solution. Therefore, we may introduce a constraint for a minimum cluster area of s^{min} for all cluster $c = 1, ..., C$ as follows:

$$s_c \geq s^{min} \quad \forall c = 1, ..., C \tag{3.16}$$

Finally, a spatial objective that is closely related to compactness, is contiguity. This objective asks for a maximization of the size of the largest cluster of each type (Cao, Huang, Wang, et al., 2012). As compactness arranges cells in preferably as least clusters as possible, it generally includes contiguity. Therefore, both aspects are encapsulated in the compactness objective.

3.3.4 Maximizing Accessibility

A good accessibility is able to increase the operational efficiency of an area and decrease its emissions. According to (Cao, Batty, Huang, et al., 2011), better accessibility planning could decrease up to 80% of the CO_2 emission resulting from human and automobile activities within city limits. In MOLA, accessibility is increased when the land-use types are close to road types that are as 'compatible' with these land-use types as possible. For example, a road meant for industrial transportation, should preferable be close to an industrial area, instead of a residential area.

Although there are many implementations of the accessibility objective, we will discuss a more-often used implementation derived from the Tongzhou

case study (Chapter 3.6). Let's say there are $r = 1, ..., R$ road types. The accessibility A_{ijk} of cell (i, j) with land-use k can be calculated as follows;

$$A_{ijk} = \sum_{r=1}^{R} e_{rij}^{k} \tag{3.17}$$

Where e_{rij}^{k} is the influence value (accessibility) of road type r for cell (i, j) of land-use type k. The e_{rij}^{k} function can differ per land-use type. In the Tongzhou case, this is for example done as follows for commercial land-use;

$$e_{rij}^{k} = (f_r^k)^{1-d} \tag{3.18}$$

And for residential and industrial land:

$$e_{rij}^{k} = f_r^k(1 - d) \tag{3.19}$$

Where d denotes the distance from cell (i, j) to a road of type r, and f_r^k is a function value that given road type r and land-use type k is defined as follows (Cao, Batty, Huang, et al., 2011):

$$f_r^k = 100 \times I_r^k \tag{3.20}$$

Here, I_r^k is the influence (compatibility) index of road type r for land-use type k. This value should be known (given) for each road and land-use type combination. This value is always in between 0 and 1, where 0 means that this road and land type are not compatible at all, and 1 means maximum compatibility.

Finally, the accessibility values A_{ijk} of all cells are summed up. A maximization of this value will lead to the best situation in terms of accessibility. The final objective function then becomes;

Maximize:

$$\sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{M} A_{ijk} x_{ijk} \tag{3.21}$$

3.3.5 Maximizing Compatibility

In reality, every type might have its preferences regarding its neighbouring land-use types (Cao, Batty, Huang, et al., 2011). For example, a residual area might prefer to be adjacent to a forest, rather than an industrial area. In that case, the residual land-use type is said to more 'compatible' with forest land-use than industrial land-use. By creating a compatibility matrix, the compatibility between different types can be determined at parcel level. (Masoomi, Mesgari, and Hamrah, 2013). This matrix contains the compatibility index of each type combination. A larger index indicates a higher compatibility, and so the more preferred they are to be neighbouring land-use types. The compatibility indices can for example be obtained from stakeholders, experts or other involved parties. Once they are obtained, the compatibility $Comp_{xy}$ between cell x and y can be calculated as follows:

$$Comp_{xy} = I_{k_x k_y}\ \alpha^1_{xy}(d_{xy}) \tag{3.22}$$

where $I_{k_x k_y}$ is the compatibility index between the land-use type of cell x (denoted as k_x) and cell y (denoted as k_y). The function $\alpha^1_{xy}(d_{xy})$ defines the effect of the distance d_{xy} between x and y and is defined as follows;

$$\alpha^1_{xy}(d_{xy}) = \begin{cases} 1 & d_{xy} \leq d^{min}_{xy} \\ \left(\frac{d^{max}_{xy} - d_{xy}}{d^{max}_{xy} - d^{min}_{xy}}\right)^{\beta} & d^{min}_{xy} \leq d_{xy} \leq d^{max}_{xy} \\ 0 & d_{xy} \geq d^{max}_{xy} \end{cases} \tag{3.23}$$

where d_{min} denotes the minimum distance and d_{max} denotes the maximum distance. The resulting objective is a maximization of the summation of the compatibility of each land unit, defined as:

Maximize:

$$\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(x_{ijk}\sum_{m=i-1}^{i+1}\sum_{n=j-1}^{j+1}\frac{Comp_{(i,j)(m,n)}}{8}\right) \tag{3.24}$$

Optionally, the objective function can be altered to stimulate a more uniform compatibility distribution over all cells. More details on these methods can be found on (Masoomi, Mesgari, and Hamrah, 2013). Finally, an identical approach to the maximization of compactness, is the minimization of the total edge length (TEL) (Schwaab, Deb, Goodman, et al., 2018). The total edge length is inversely related to compactness. Note, that the total edge length can only be calculated using a four-neighbour approach.

An alternative for maximizing 'compatibility' is maximizing 'dependency'. In that case, a land-use is said to depend on other land-use types in its vicinity (Masoomi, Mesgari, and Hamrah, 2013). However, two land-use types that depend on each other, could also be seen as two land-use types that are highly compatible with each other. As such, the dependency objective is somehow equivalent to the compatibility objective, and can be implemented in a similar manner as the compatibility objective.

3.4 Optimization Approaches

There are a variety of approaches for optimizing multi-objective problems (Amine, 2019). In the following sections, several approaches in the context of MOLA will be handled. These can be classified into a priori and a posteriori approaches.

3.4.1 A Priori

When using an a priori approach, also called 'decide-then-search', the decision maker transforms the mathematical multi-objective model into a mono-objective one before searching for the solution(s) (Matthews, Craw, Elder, et

al., 2000). Several methods have been introduced from this perspective. In this section we will discuss the two most frequently used a priori methods in the domain of MOLA, namely, the weighted sum method and goal programming (Amine, 2019).

The weighted sum method is the most simple approach based on aggregation (Cao, Huang, Wang, et al., 2012). For this approach, a weight is given to each objective, after which they can all be combined to one single objective to minimize. For the multi-objective land-use allocation problem with objectives $o = 1, ..., O$, this would result in the following single-objective function;

Minimize:

$$f_{total} = -\sum_{o=1}^{O}\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{M}\alpha_o B_{oijk} x_{ijk} \tag{3.25}$$

where B_{oijk} is the parameter based on cell (i, j) for type k of objective o and α_o equals the weight of objective o.

However, the weighed sum method has several disadvantages. First of all, it can be difficult for decision makers or planners to value or weight the objectives directly (Cao, Huang, Wang, et al., 2012). This is particularly true, when each objective value has a different scale. In certain situations, particularly where there is conflict over a decision, it may be impossible to agree a priori weightings or orderings (Matthews, Craw, Elder, et al., 2000). Furthermore, it has been shown that the use of linear forms such as the above can lead to highly biased results (Stewart, Janssen, and Herwijnen, 2004). Resulting in, for instance, some objectives being optimized very badly compared to others. In real-world land-use planning scenarios, such solution properties would often be unacceptable.

Due to the scaling difficulties of the weighted sum approach mentioned above, a more often used approach is some general form of goal programming (GP). An often-used goal programming approaches in the spatial optimization domain, is the Chebyshev approach. The approach, also called the Minimax approach, was first introduced in (Flavell, 1976). The Chebyshev approach minimizes the maximum weighted deviation of any function from a certain reference point (M.C. Roberts and Vaughan, 2012). A slightly modified version of the Chebyshev goal programming approach was applied to MOLA in (Cao, Huang, Wang, et al., 2012). For objectives $i = o, ...O$, this approach was defined as follows;

Minimize:

$$f_{total} = \sum_{o=1}^{O}\alpha_o\left(\frac{f_o - f_o^{min}}{f_o^{max} - f_o^{min}}\right) \tag{3.26}$$

where f_o is the objective value, f_o^{min} is the ideal value, f_o^{max} is the worst value and α_o is the weight given to objective o. This approach addresses the scale differences of each objective value while helping planners capture preferences for different objectives (Mohammadi, Nastaran, and Sahebgharani, 2016).

The approach of (Cao, Huang, Wang, et al., 2012) still requires planners or stakeholders to define weights for each objective. In case this is considered to be unwanted, a weight-free approach can be used, as was shown in (Aerts, Herwijnen, and Stewart, 2003). It is defined as follows:

Minimize:

$$f_{total} = \sum_{o=1}^{O} \left(\frac{f_o - f_o^{min}}{f_o^{max} - f_o^{min}} \right)^{\rho} \tag{3.27}$$

where ρ should be a suitably large power. The use of ρ places more weight on the least well satisfied goal, aiming to create a more uniform minimization of all objective functions.

Besides the objectives being optimized, the solution also has to meet constraints regarding the number of cells allocated to each cell. One can either formulate these constraints as hard constraints, or handle them as goals as well. In the second case, the constraint goals have to be added to the function value as well, in order to penalize a violation. An example of how this can be implemented, was shown in (Stewart, Janssen, and Herwijnen, 2004). Lets say that for each number of cells allocated to a type S_k, there is an upperbound U_k and lowerbound L_k as specified earlier in Equation (3.6). Also, lets assume that we have a minimum cluster size s^{min} for the size s_c of all clusters $c, ..., C$ as shown in Equation (3.16). The a priori goal programming objective of Equation (3.27), can then be extended by adding the following three terms (Stewart, Janssen, and Herwijnen, 2004). For the lower bound L_k on S_k:

Minimize:

$$\left(\frac{max(0, L_k - S_k)}{\beta_k^0} \right)^{\rho} \tag{3.28}$$

For the upper bound U_k on S_k:

Minimize:

$$\left(\frac{max(0, S_k - U_k)}{\beta_k^0} \right)^{\rho} \tag{3.29}$$

And for the minimum cluster size s^{min} of s_c:

Minimize:

$$\left(\frac{max(0, s_{min} - s_c)}{\beta_k^1} \right)^{\rho} \tag{3.30}$$

where β_k^0 denotes the scaling factor relating to the upper and lower bounds, β_k^1 the scaling factor of the minimum cluster size and ρ the power used for

goal programming as used in Equation (3.27).

3.4.2 A Posteriori

A posteriori methods, also called search-and-decide, first search for the Pareto set of solutions and then present these to the decision maker (Matthews, Craw, Elder, et al., 2000). In general, posteriori methods are preferred and found superior for the MOLA problem. It does not yield the disadvantages stated in Section 3.4.1, and presents the conflicts between objectives without (the need for) prior knowledge of the solution space. Yet, such a construction is often more difficult and computationally consuming. Therefore, approximation algorithms are commonly suggested in this case (Amine, 2019). They consist in constructing a subset of the Pareto set, or otherwise a set of efficiently near-Pareto optimal solutions. The construction of the actual Pareto set or an approximating of it, is referred to as Pareto optimisation.

A posteriori approaches can also be combined with a priori approaches to create a so-called progressive approach. This is a hybridisation of a posteriori and a priori, where the decision-maker provides a kind of guidance to the algorithm execution. It often consists of an a priori approach, that learns from the user's preferences among intermediate a posteriori results, for as long as the algorithm runs (Amine, 2019).

3.5 Problem Representations

The MOLA problem can be represented in different ways. The representation used, can influence the efficiency and performance of the optimisation approach or algorithms being used. The problem definition and objectives described in Chapter 3.2 and 3.3 make use of the grid representation. In this chapter, several possible representations including their advantages and disadvantages are outlined.

3.5.1 Grid Representation

The grid representation is the simplest and most commonly used representation (Matthews, Craw, Elder, et al., 2000). With the grid representation, a solution is encoded as a two dimensional array in which each value corresponds to a land-use type. Each cell in the array hereby represents a geographic area of fixed size. The size of these cells determines the level of detail and the eventual size of the data structure.

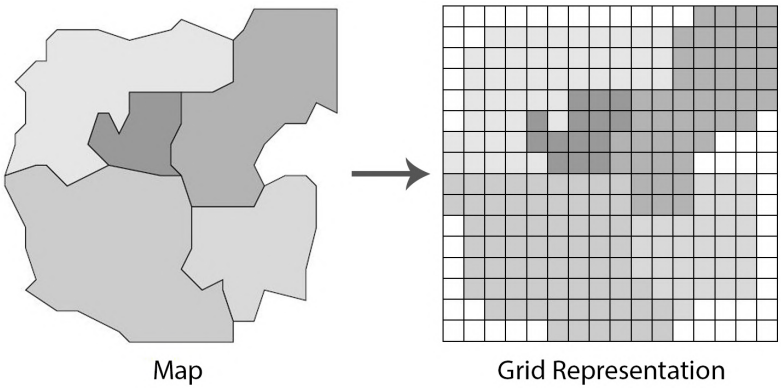


FIGURE 3.2: Grid Representation (Matthews, Craw, Elder, et al., 2000)

The main advantages of the grid representation is that is a simple structure that can easily be manipulated. However, the grid representation also has a significant downside, which is redundancy. This is particularly likely when using categorical data such as land-use types on predefined parcels (Matthews, Craw, Elder, et al., 2000). For example, look at the grid representation of a certain map in Figure 3.2. The actual map exists out of 5 parcels, to which we need to assign land-use types. When using a grid representation of 16 x 16, this turns into a double array of 256 cells. Meaning, we now have to allocate land-use types to 256 cells, while in reality we are only allocating a type to 5 parcels. Eventually, the results of the grid can be translated back to land-use types for each parcel, by for each parcel selecting the most occurring land-use type of its overlapping cells in the grid. As can be seen, the grid representation would deliver a lot of redundancy, resulting in more overhead in the optimization process. However, when there are no strict parcels, the grid representation can be a suitable approach. In that case, this representations allows for an easy creation and modification of clusters of all kinds of shapes, in contrast to for example vector representations that will be discussed later.

3.5.2 Quad-Tree Representation

In order to address the redundancy problem of the grid representation, other data structures have been proposed. One of these structures, is the quad-tree structure. The quad-tree recursively subdivides space into progressively smaller spatial units, indexed using a tree-structure (Matthews, Craw, Elder, et al., 2000). This subdivision continues, until either a node in the tree is homogeneous with only one land-use represented or the maximum recursion depth is reached.

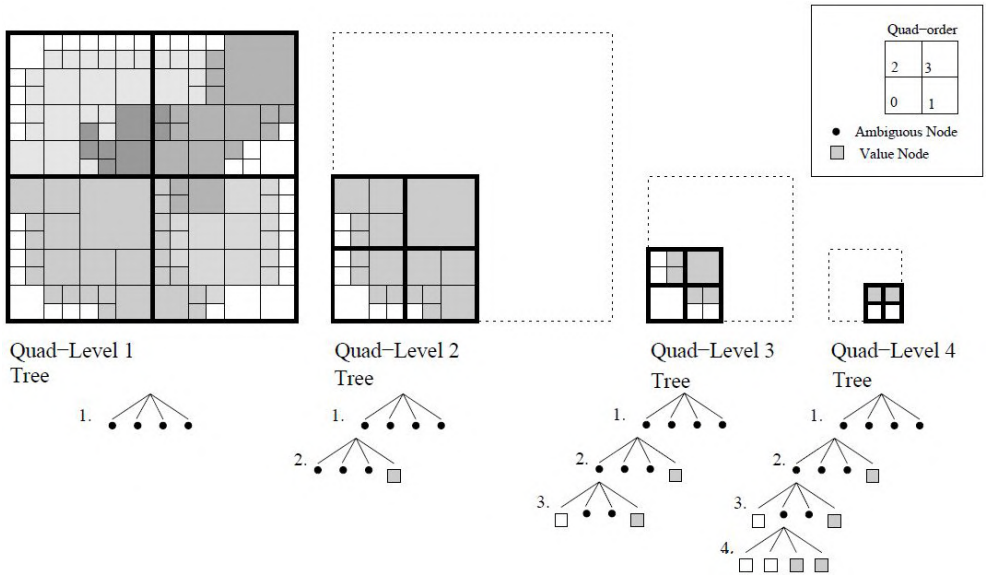


FIGURE 3.3: Quad-Tree Representation (Matthews, Craw, Elder, et al., 2000)

In Figure 3.3, four levels of recursion are shown on the same polygon map as in Figure 3.2. As can be seen, the grid is continuously split in four separate grids. Redundancy is reduced when one split grid completely exists out of one color.

The advantage of the quad-tree structure is that is able to decrease part of the redundancy of the grid structure. However, this comes at the expense of algorithmic and computational complexity (Matthews, Craw, Elder, et al., 2000). Besides that, building and repairing the quad-tree structure could bring more overhead to the optimisation process.

3.5.3 Polygon Vector Representation

A different representation type is based upon vectors. The polygon-level vector-based representation, is the most-used vector representation. It is also called the fixed-length vector or land block representation (Datta, Deb, Fonseca, et al., 2007). The representation directly maps the land-uses of individual fields to a vector, as shown in Figure 3.4. Similar to the grid-based approach, each value is again a vector of size K, of which the value at index $k = 1, ...K$ is 1 in case type k is given to it (0 otherwise). The major advantage of this representation is that it is not bothered by any redundancy and scales well with an increasing number of parcels. As such, a vector representations is able to decrease the size of the decision variables. (Schwaab, Deb, Goodman, et al., 2018). Still, this representation also has several downsides. First, it is more difficult to combine, split or modify parcel shapes than when using the patch-level representation. Secondly, a one-value fixed-length vector does not maintain spatial data which can complicate the working with spatial objectives and operators. As such, it may require the incorporation of supplementary algorithmic schemes or a GIS system to take note of this data. (Cao, Batty, Huang, et al., 2011).

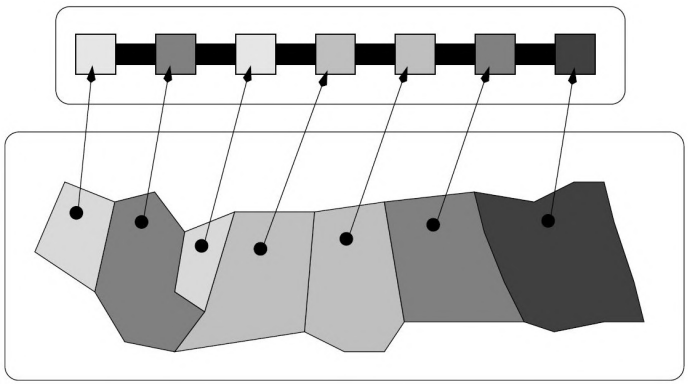


FIGURE 3.4: Polygon-Level Vector Representation (Matthews, Craw, Elder, et al., 2000)

The fixed-length vector representation can be formulated as follows. Let us assume that the area exists out of $i = 1, ..., N$ vectors with $k = 1, ..., K$ types. The variable $x_i k$ is 1 if the vector i equals k, and 0 otherwise. B_{ik} equals the parameter of an objective, of which its value differs per location. For every objective, the fixed-length MOLA model can be formulated as follows (Cao and Ye, 2013):

Minimize:

$$-\sum_{k=1}^{K}\sum_{i=1}^{N} B_{ik}x_{ik} \tag{3.31}$$

Subject to:

$$\sum_{k=1}^{K} x_{ik} = 1 \quad \forall i = 1, ..., N \tag{3.32}$$

where:

$$x_{ik} \in \{0, 1\} \quad \forall i = 1, ..., N, k = 1, ..., K \tag{3.33}$$

Note, that most constraints mentioned before, such as the boundary constraints of Equation (3.7), would now require a reformulation as well.

Another polygon-level vector-based representation is the variable-length vector (also called percentage and priority) representation (Datta, Deb, Fonseca, et al., 2007). This representation is more complex, and focuses on the 'percentage and priority' in the allocation of land-use. It therefore requires more a priori knowledge of the scenario. The exact definition and working of this structure can be found in (Matthews, Craw, Elder, et al., 2000).

3.5.4 Patch Vector Representation

In previous methods, the land-use type of land parcels were either allocated cell by cell or by a single value in a polygon-based vector. The polygon-based representation had a low redundancy, but brought difficulties when modifications of the parcels are allowed. The cell by cell approach had a high redundancy, but parcel shapes could be modified easily. The patch-level vector

representation, brings a balance between both (Liu, Peng, Jiao, et al., 2016). It looks at the map from a patch-level, which is simply a cluster of cells from the same land-use type. To decrease the number of spatial units (and hence the computational effort), we first transform the input raster map representing the current status of land-use into a patch or cluster map, where neighboring raster cells of the same type are aggregated (Strauch, Cord, Pätzold, et al., 2019). Next, a solution can be represented as a vector, where the index of the value corresponds to the ID of the patch in the patch ID map. This is shown in Figure 3.5. In case the parcels are not modified, the same patch ID map can be used for multiple solutions. In case the parcels are modified, a new patch ID map can be recreated or the current one can be modified. Note, that the last land-use type can be left out of the vector, as this can be allocated to all remaining patches.

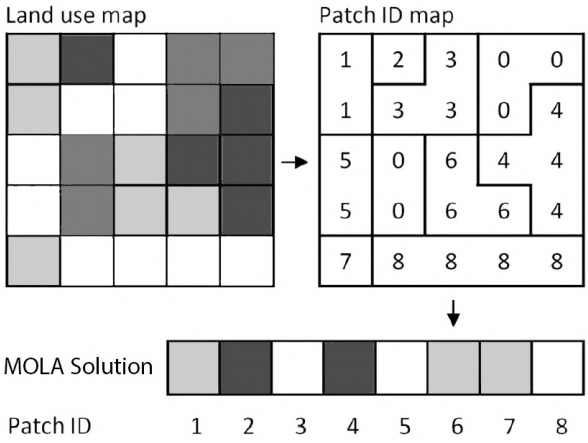


FIGURE 3.5: Patch-Level Vector Representation (Strauch, Cord, Pätzold, et al., 2019)

The patch-level representation decreases the redundancy of a single solution. As such, it is more efficient to work with when handling multiple different solutions and is better of handling larger areas than cell-based representations (Liu, Peng, Jiao, et al., 2016). Another big advantage, is that in case the user would like to do operations/calculations on patches (instead of cell), this can be done much more efficiently. However, it might become expensive if the ID patch map needs to be recreated regularly. In that case, it might become more efficient to use the simple grid representation again.

3.5.5 Multi-Dimensional Representation

An extra dimension can be added to any of the previously described structures, to include the changes in land-use over time. (Datta, Deb, Fonseca, et al., 2007). In case of the vector representations, this would be a second dimension. In case of a grid representation, this would be a third dimension. For now, we will elaborate on the third-dimensional grid representation. Each cell would now obtain a vector of land-use allocation over time. An example is shown in Figure 3.6. Here, e_ij contains the land-use vector y of cell (i, j). The vector y now contains $1, ..., n$ values for year 1 to n on the t-th axis.

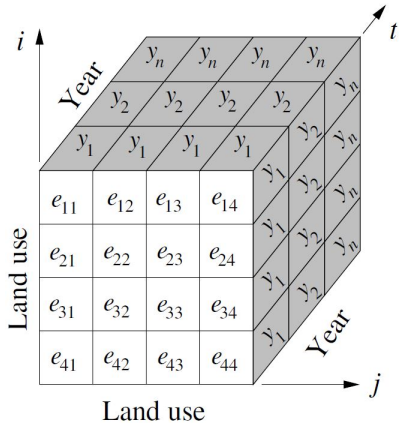


FIGURE 3.6: Model Of A 3-Dimensional Landscape (Datta, Deb, Fonseca, et al., 2007)

3.6 Example Case Studies

Different case studies have been introduced and optimized throughout previous researches. In this chapter, we will give a short overview of these case studies and discuss the two most used case studies in more detail.

3.6.1 Overview Case Studies

In Table 3.1, ten MOLA use cases are listed, including their number of different land-use types, objectives and area size. First of all, there are several relatively small cases, such as the Baboldash case of 0.2 km^2 (Mohammadi, Nastaran, and Sahebgharani, 2016), the Jisperveld case of 4 km^2 (Aerts, Van Herwijnen, Janssen, et al., 2005) and the As Pontes case of 25km^2 (Aerts, Eisinger, Heuvelink, et al., 2003). Secondly, there are medium sized cases, like the Tonghzou case of 906 km^2 (Cao, Huang, Wang, et al., 2012) and the Panyu case of 786 km^2 (Liu, Li, Shi, et al., 2012). Thirdly, we have the relatively large cases, such as the RDCO case of 2902 km^2 (Li and Parrott, 2016) and the Terra Cha case of 1832 km^2 (Santé-Riveira, Boullón-Magán, Crecente-Maseda, et al., 2008). The latter has up to 13 different land-use types. Finally, the exact area size of three cases are unknown. Those are the Baixo Alentejo case (Datta, Deb, Fonseca, et al., 2007), the District 1 case of Tehran (Masoomi, Mesgari, and Hamrah, 2013) and the Guitiriz case (Porta, Parapar, Doallo, et al., 2013).

TABLE 3.1: Specifications of several MOLA case studies.

Case Study	Land-Use Types	Objectives	Area Size
As Pontes, Spain	3	2	25 km^2
Jisperveld, Netherlands	9	6	4 km^2
Baixo Alentejo, Portugal	5	3	Unknown
Terra Cha, Spain	13	3	1832 km^2
Tongzhou, China	6	8	906 km^2
Panyu, China	7	3	786 km^2
District 1, Region 7, Tehran	12	4	Unknown
Guitiriz, Spain	4	2	Unknown
Baboldasht, Iran	7	5	0.2 km^2
RDCO, Canada	8	3	2902 km^2

The two most used case studies, are the Jisperveld case study and the Tonghzou case study. As such, these will discussed in more detail in Chapter 3.6.2 and 3.6.3. For more details on any of the other previously named case studies, view the associated research.

3.6.2 Case 1: Jisperveld, The Netherlands

The first case study that will be discussed, is the management of the Jisperveld region in The Netherlands. The Jisperveld is a natural area in the Netherlands of 2000ha. (Stewart, Janssen, and Herwijnen, 2004). It is the breeding ground of several rare birds, and so 800ha of this area belongs to a natural conservation organization. The future land-use planning and management has led to a discussion between different stakeholders, including agricultural, recreational and natural parties (Aerts, Van Herwijnen, Janssen, et al., 2005) (Aerts, Herwijnen, and Stewart, 2003). Every party has its own vision on how the Jisperveld should allocate its land-use. Therefore, the Jisperveld has been translated into an optimizable MOLA problem to support these parties in coming up with a plan.

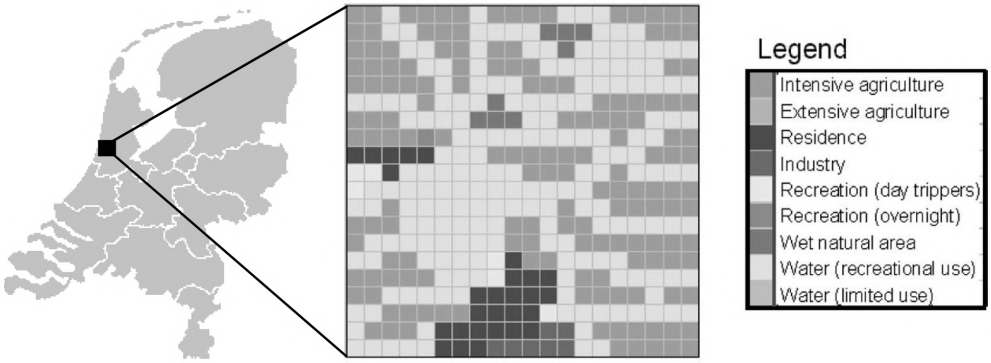


FIGURE 3.7: Jisperveld (Aerts, Van Herwijnen, Janssen, et al., 2005)

For purposes of the present numerical studies, a 400ha region was selected and displayed in the form of a 2020 grid in Figure 3.7 (Stewart, Janssen, and Herwijnen, 2004). As many as 33 distinct land-use types could be identified in this area, but for purposes of the illustration these have been reduced to seven. The legend in Figure 3.7 identifies nine land-use types, which includes the two possible future types “water (limited access)” and “extensive agriculture". These do not occur in the current situation. Together with the stakeholders, a total of six different objectives have been set up (Aerts, Van Herwijnen, Janssen, et al., 2005):

1. Maximize the natural value of the area.
2. Maximize the recreational value of the area.
3. Minimize the cost of changing land-use.
4. Minimize the number of clusters.
5. Maximize the cluster size.
6. Maximize the compactness.

Note, that the last three objectives are all similar to the compactness objective of Equation (3.13). The values of objective 1 and 2 can be calculated using the maximization of Equation (3.9). For a land-use type k, the value of C_{ijk} is either a uniform value (the same for each cell (i, j)), or a variable value (different per cell (i, j)). These values are shown in Table 3.2; all values are scaled between 1 and 10 and the variable values are shown as 'Map'.

TABLE 3.2: Cost values C_{ijk} of Objective 1 (Nature Value) and Objective 2 (Recreation Value) (Stewart, Janssen, and Herwijnen, 2004)

Land-Use Type (k)	Nature Value	Recreation Value
1. Intensive agriculture	4	6
2. Extensive agriculture	Map	Map
3. Residence	3	3
4. Industry	1	1
5. Recreation (day trippers)	5	Map
6. Recreation (overnight)	5	Map
7. Wet natural area	Map	7
8. Water (recreational use)	7	Map
9. Water (limited access)	Map	1

Furthermore, the costs of adjusting a present type k_c into a new type k_f are shown in Table 3.3 (Aerts, Herwijnen, and Stewart, 2003). These can be used to calculate the value of Objective 3, using Equation 3.9 with the transition costs as C_{ijk}. Note, that not all land-use type combinations in Table 3.3 have associated transition costs (noted with a '-'). These types are simply not allowed to be converted into each other, and are fixed. In Figure 3.8, all cells with a fixed land-use type are visualized.

TABLE 3.3: Transition Costs Between Land-Use Types. Forbidden transitions are noted with '-' (Aerts, Herwijnen, and Stewart, 2003)

	Current Land-Use Type (k)								
Future Land-Use Type (k)	1	2	3	4	5	6	7	8	9
1	0	1000	1000	500	-	7000	-	-	-
2	-	0	-	-	-	-	-	-	-
3	-	-	0	-	-	-	-	-	-
4	-	-	-	0	-	-	-	-	-
5	-	-	9000	-	0	5000	-	-	-
6	-	-	-	-	-	0	-	-	-
7	-	-	-	-	-	-	0	-	-
8	-	-	-	-	-	-	-	0	1000
9	-	-	-	-	-	-	-	-	0

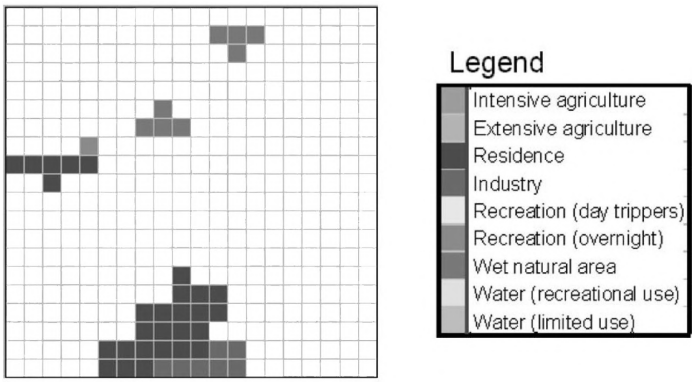


FIGURE 3.8: Map With The Current Fixed Types (Aerts, Herwijnen, and Stewart, 2003)

Finally, there are several constraints to which the solution should obey. These are a lower and upper bound on the number of cells per land-use type (Equation (3.6)), and a minimum cluster size per land-use type (Equation (3.16)) (Aerts, Herwijnen, and Stewart, 2003). The values of these constraints are listed in Figure 3.4. Note, that certain land-use types (such as Intensive Agriculture) need to decrease in area size, whereas other land-use types (such as Extensive Agriculture) need to increase. Also, different minimum cluster sizes are given for each land-use type.

TABLE 3.4: Constraint Vaues per Land-Use Type (Stewart, Janssen, and Herwijnen, 2004)

Land-Use Type (k)	Lower Bound	Upper Bound	Current	Min. Cluster Size
1	100	130	157	4
2	27	57	0	3
3	28	35	28	3
4	5	9	7	2
5	3	10	6	3
6	1	5	1	1
7	4	20	8	3
8	150	193	193	4
9	0	43	0	4

3.6.3 Case 2: Tongzhou New Town, China

In the previous years, China has been experiencing an enormous industrial and urban growth. One of the most rapidly developing areas of China is Tongzhou (Cao and Ye, 2013). This area is located south of Beijing and covers 906 km^2. The area contains 11 towns and 4 communities, and has always been involved in a debate on how the area can best be further developed in the future (Cao, Batty, Huang, et al., 2011). Especially the Tonghzou New Town district, shown in Figure 3.9, asked for a lot of discussions. In order to offer help, the Tongzhou New Town region has been converted into an optimizable MOLA problem, which can be used to obtain possible favorable future scenarios.

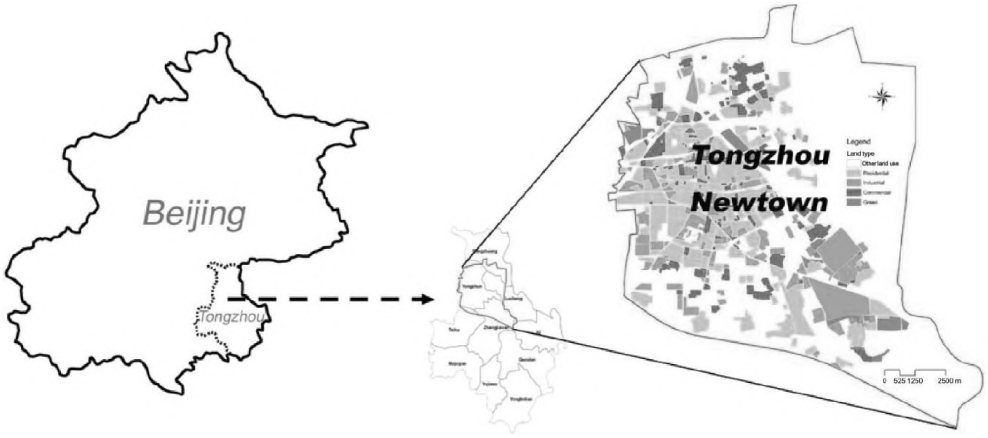


FIGURE 3.9: The Tongzhou New Town District (Cao and Ye, 2013)

The map of Tongzhou New Town includes five different land-use types: residential, industrial, commercial, green and undeveloped land. Different representations can be used, such as a vector of size 586 (Cao and Ye, 2013) a small grid with cells of 400mx400m (Cao, Batty, Huang, et al., 2011) or a large grid with cells of 100mx100m (Cao, Huang, Wang, et al., 2012). For now, we will focus on the large grid representation, as is shown in Figure 3.10.

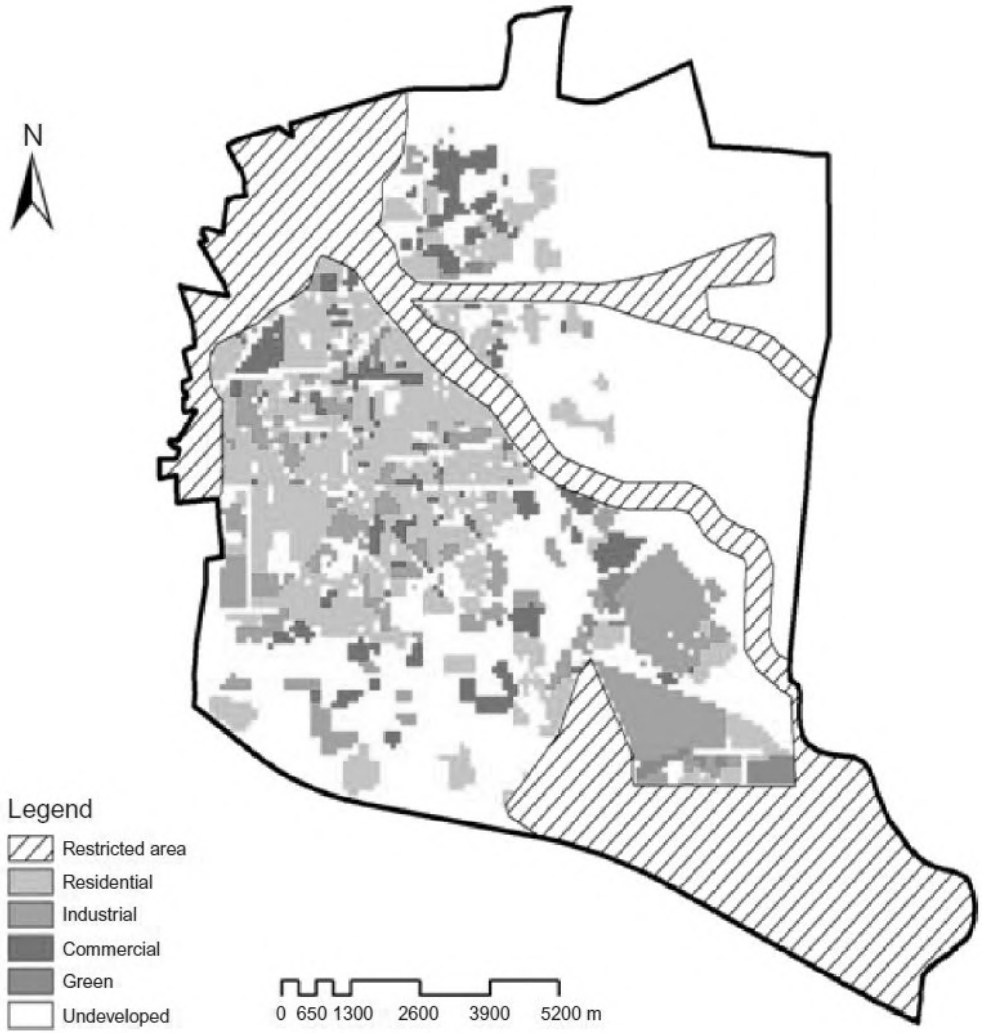


FIGURE 3.10: Current Tongzhou New Town Land-Use Map (Cell Size 100m x 100m) (Cao, Batty, Huang, et al., 2011)

Many objectives can, if desired, be included in the optimization of the Tongzhou case study. For now, we will shortly discuss eight different objectives that can be optimized. For more details, view (Cao, Huang, Wang, et al., 2012). The following objectives will be discussed;

1. Maximize the gross domestic product (GDP).
2. Minimize the conversion.
3. Maximize the geological suitability.
4. Maximize the ecological suitability.
5. Maximize the accessibility.
6. Minimize the not in my back yard (NIMBY) factor.
7. Maximize the compactness.
8. Maximize the compatibility.

At first, the maximization of the gross domestic product (GDP). The GDP can be used to evaluate land-use scenarios from an economic standpoint. Using historical data and statistical methods, the correlation between these land-use types can be obtained, which can be used to represent the anticipated GDP in 2020 (Cao, Huang, Wang, et al., 2012). The resulting GDP ratios of different land-use types are shown in Figure 3.5. The objective value can now be

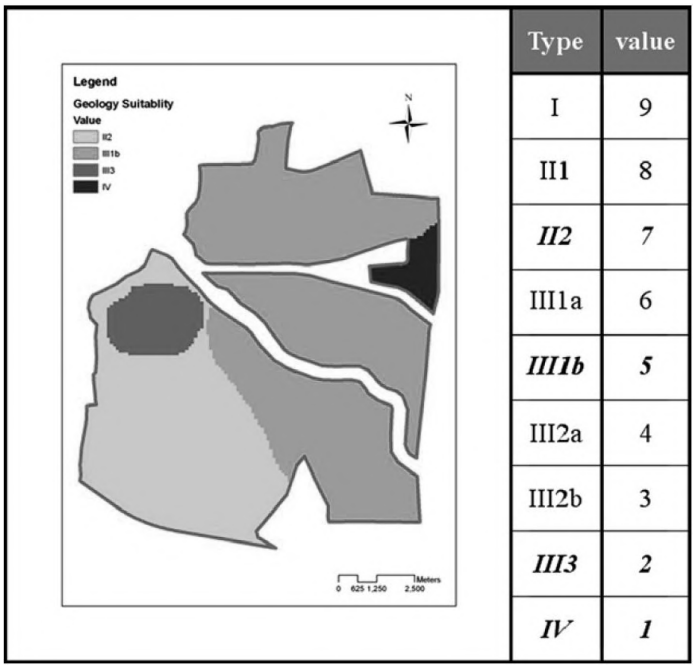
calculated using Equation (3.9), where C_{ijk} is equal to the GDP ratio of k, regardless of i and j.

TABLE 3.5: The GDP Ratio Per Land-Use Type (Cao, Huang, Wang, et al., 2012)

Land-Use Type	GDP Ratio
Residential	0
Industrial	59.92
Commercial	505.20
Green	0
Undeveloped	0

The second objective, is the minimization of the conversion. Conversion costs for different land-uses decreases the expenditure of the social capital, while enhancing the economic benefit of the society. Evidently, the cost of different land-use types is different. However, under the consideration of data limitation and the inaccuracy by the Delphi method, the minimization of conversion can be simplified to the minimization of the total number of land-use changes (Cao, Batty, Huang, et al., 2011).

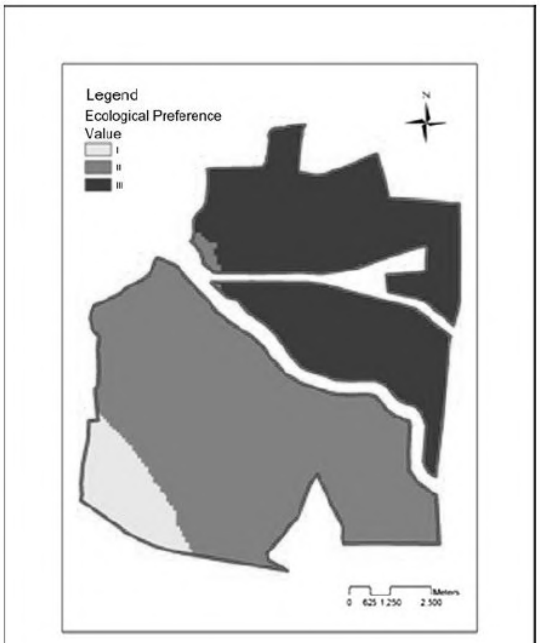
The third objective, is the maximization of the geological suitability. Geological suitability states how 'suitable' a cell is for a land-use type different than 'undeveloped'. Figure 3.11 shows the geological suitability map of Tongzhou. The districts for geological construction have a trend of degradation from I to IV. Hence, the maximization of the sum of all the cells' suitability values of all cells other than 'undeveloped' can bring about a better solution (Cao, Batty, Huang, et al., 2011).



Type	value
I	9
II1	8
II2	*7*
III1a	6
III1b	***5***
III2a	4
III2b	3
III3	*2*
IV	*1*

FIGURE 3.11: Geology Suitability Values Per Region (Cao, Huang, Wang, et al., 2012)

The fourth objective, is the maximization of the ecological suitability. Ecological factors need to be incorporated as part of sustainable land-use allocation. Based upon the values of the world's ecosystem services and natural capital, including the suitability of allocating green land per region, the values of Figure 3.12 arose. As shown, allocating green in district II or III adds up more to the objective value than other land-use types. Since district I is highly unsuitable for green land-use, other types are preferred here instead. The sum of all suitability values results in the final objective value to be maximized.



Ecological Value	Green	Others
I	1	2
II	2	1
III	3	0

FIGURE 3.12: Ecological Suitability Values Per Region (Cao, Huang, Wang, et al., 2012)

The fifth objective, holds a maximization of the accessibility. The accessibility objective was described earlier in Chapter 3.3.4. Its objective value is calculated according to Equation (3.17), (3.18), (3.19), (3.20) and (3.21). In the Tongzhou case study, there are three different road types; roads for living, roads for transportation and roads for mixed use (both). The influence (compatibility) index I_r^k of each road type r for every land-use type k is shown in Table 3.6. Finally, the road network of Tongzhou is shown in Figure 3.13.

TABLE 3.6: Influence Index for Different Roads (Cao, Huang, Wang, et al., 2012)

	Residential	Industrial	Commercial
Main Road for Living	1	0.7	0.875
Main Road for Transportation	0.7	1	0.7
Main Road for Mixed Use	0.875	0.875	1

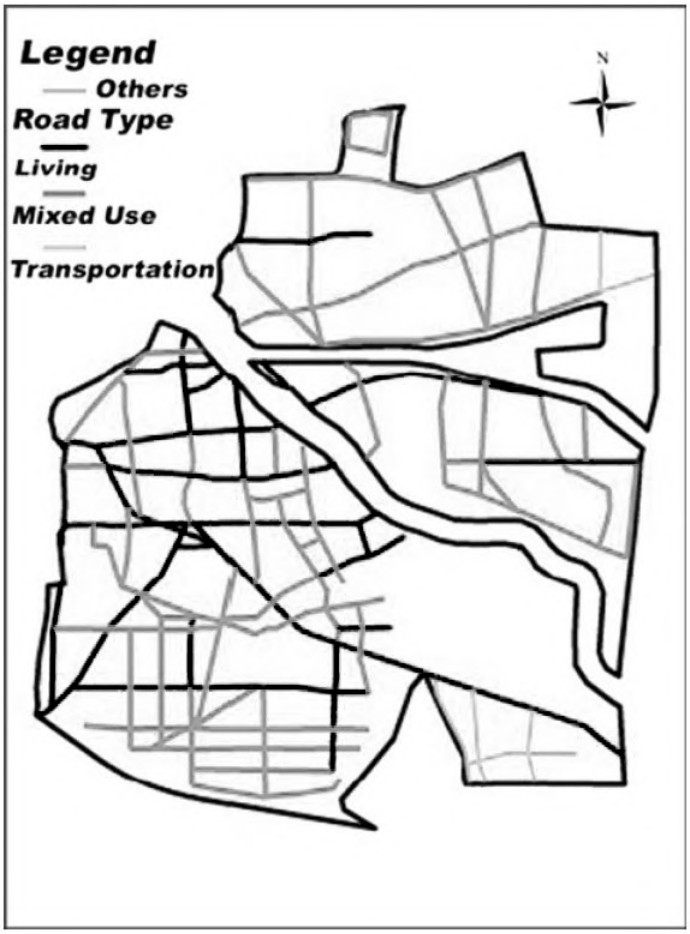


FIGURE 3.13: Roads Network of Tongzhou (Cao, Huang, Wang, et al., 2012)

The sixth objective, is the minimization of the NIMBY influence. The term of NIMBY (Not in My Back Yard) is used to refer to the opposition by residents to new developments in their proximity (Cao, Huang, Wang, et al., 2012). It may not be easy to change the location of projects such as railways, landfill fields, or power stations, etc. However, this study takes a novel attempt to address the NIMBY factor. Owing to data limitation, only railways are considered in the Tongzhou case study. More details on this objective, including a Euclidean distance based function decreasing map, can be found in (Cao, Huang, Wang, et al., 2012). The purpose of this objective is to minimize the occurrence of residential and commercial land inside the high influence value of railways in the city.

The final two objectives, are the maximization of compactness and compatibility. The first objective is simply calculated using the standard eight-neighbor method of Equation (3.13). The second is calculated using the compatibility

function of Equation (3.24). The compatibility value $Comp_{xy}$ between cell x and y depends on the land-use type of both cells. The value for each possible combination in the Tongzhou case, is given in Table 3.7

TABLE 3.7: Compatibility Values in Tongzhou (Cao, Huang, Wang, et al., 2012)

	Residential	Industrial	Commercial	Green	Undev.
Residential	1				
Industrial	0.41	1			
Commercial	0.95	0.48	1		
Green	1	0.88	0.62	1	
Undeveloped	0.47	0.75	0.41	0.74	1

Finally, several constraints have been set up. First of all, there are several restricted areas that can not be changed. These areas are visualised in Figure 3.10 (Cao, Batty, Huang, et al., 2011). Finally, a minimum bound was set on the number of cells to which the residential land-use type should be allocated. Based upon future estimations, this resulted in a minimum bound of 3150 cells (when using a cell size of 100m x 100m) (Cao and Ye, 2013).

Chapter 4

Previous Research

4.1 Linear Programming

One of the classic techniques for combinatorial optimisation is linear programming (LP) (Stewart, Janssen, and Herwijnen, 2004). It aims to optimise a single linear objective function, given multiple linear equality and inequality constraints. Over the years, linear programming solvers have become faster and more sophisticated, allowing an optimisation problem to be solved efficiently when a single clear objective could be identified. Linear programming can be used to optimize multi-objective problems too, by converting the objectives into one single objective. This can for example be done by using aggregation procedures such as weighted sum as discussed in Chapter 3.4.1, in which all objectives are weighted and summed up (Stewart, Janssen, and Herwijnen, 2004).

In the research of (Aerts, Eisinger, Heuvelink, et al., 2003), several LPs were set up for solving the multi-objective land-use allocation problem with two objective functions; minimizing the developments costs as well as maximizing the compactness. The objectives were combined using the weighted sum method of Equation (3.25). The LP was then defined as (Aerts, Eisinger, Heuvelink, et al., 2003):

Minimize:

$$\sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{M} C_{ijk}x_{ijk} - \alpha \sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ijk} \tag{4.1}$$

Subject to:

$$y_{ijk} \leq 4x_{ijk} \tag{4.2}$$
$$y_{ijk} \leq x_{i-1jk} + x_{i+1jk} + x_{ij-1k} + x_{ij+1k} \tag{4.3}$$
$$y_{ijk} \geq x_{i-1jk} + x_{i+1jk} + x_{ij-1k} + x_{ij+1k} - 4(1 - x_{ijk}) \tag{4.4}$$
$$y_{ijk} \geq 0 \tag{4.5}$$
$$\forall k = 1, ..., K, i = 1, ..., N, j = 1, ..., M$$

including Equation (3.2), (3.3) and (3.33). Hereby, C_{ijk} are the development costs of allocating land-use type k to cell (i, j) where α equals the (relative) weight of the compactness objective. Constraints (4.2), (4.3), (4.4), (4.5) are necessary to implement the compactness objective function while still maintaining a linear problem. Eventually, (Aerts, Eisinger, Heuvelink, et al., 2003) also introduces the concept of buffer-cells to encourage compactness. We will

not elaborate on these constraints.

The linear programming model of (Aerts, Eisinger, Heuvelink, et al., 2003) was eventually tested on a case study of 300x300 cells. Unfortunately, it was impossible to solve the whole area within one model run; it was too complicated and did not finish (within a reasonable amount of time). Therefore, the area was split up to sample areas of 30x30 and even 16x16, after which the results for each area were combined. Since most multi-objective problems tend to have a large number or cells, this turned out to be a major drawback of the linear programming approach. Also, the model scaled badly for an increase in the number of objectives. When solving for the development costs only, it only took about two minutes for a 30x30 grid. When the compactness objective was added, this increased to more than eight hours. Since most MOLA problems use to have more than two objectives, this is expected to be a major bottleneck. As such, linear programming is considered not to be a suitable approach for realistic MOLA problems. After the research of (Aerts, Eisinger, Heuvelink, et al., 2003), the interest in LP for MOLA therefore significantly decreased.

4.2 Local Search Heuristics

Given the limitations of linear programming, heuristic local search methods have been examined to handle the MOLA problem. In this chapter, we will discuss approaches based upon two heuristics; simulated annealing and tabu search.

4.2.1 Simulated Annealing

The search heuristic that has been applied to MOLA in most researches, is simulated annealing. The simulated annealing algorithm emulates the behaviour of a thermodynamic system. It applies local search, with an acceptance-function that is based upon a temperature variable s_0 (Santé-Riveira, Boullón-Magán, Crecente-Maseda, et al., 2008). As the temperature decreases, the probability of accepting a new worse solution found by local search decreases as well. The general procedure of (single-objective) simulated annealing works at follows; At first, an initial solution is generated. Next, a new solution is created out of this solution, using a local search operator. In case the objective value of this solution is an improvement over the objective value of the current solution, this solution becomes the new current solution. In case it is worse, it can still become the new current solution with the following probability (Aerts, Herwijnen, and Stewart, 2003);

$$P(acceptchange) = exp(\frac{f(0) - f(1)}{s_0}) \tag{4.6}$$

where s_0 is the temperature, $f(0)$ equals the objective value regarding the current solution and $f(1)$ equals the objective value regarding the new solution. This procedure continues, until a certain criterion is met. Note, that the chance of accepting a worse solution reduces as the temperature s_0 decreases. Usually, the temperature is decreased using a constant multiplication factor r (Aerts, Herwijnen, and Stewart, 2003);

$$s_{i+1} = r * s_i \tag{4.7}$$

where $0 < r < 1$ and i denotes the iteration index.

The standard simulated annealing algorithm is meant for single-objective optimization problems. In (Aerts, Herwijnen, and Stewart, 2003) goal programming was applied to aggregate the objectives into a single objective, as discussed earlier in Chapter 3.4.1. The resulting simulated annealing algorithm for MOLA problems is shown in Figure 4.1. New solutions are being generated by swapping the land-use type of two randomly selected cells. A maximum number of iterations L_m was set as the termination criterion.

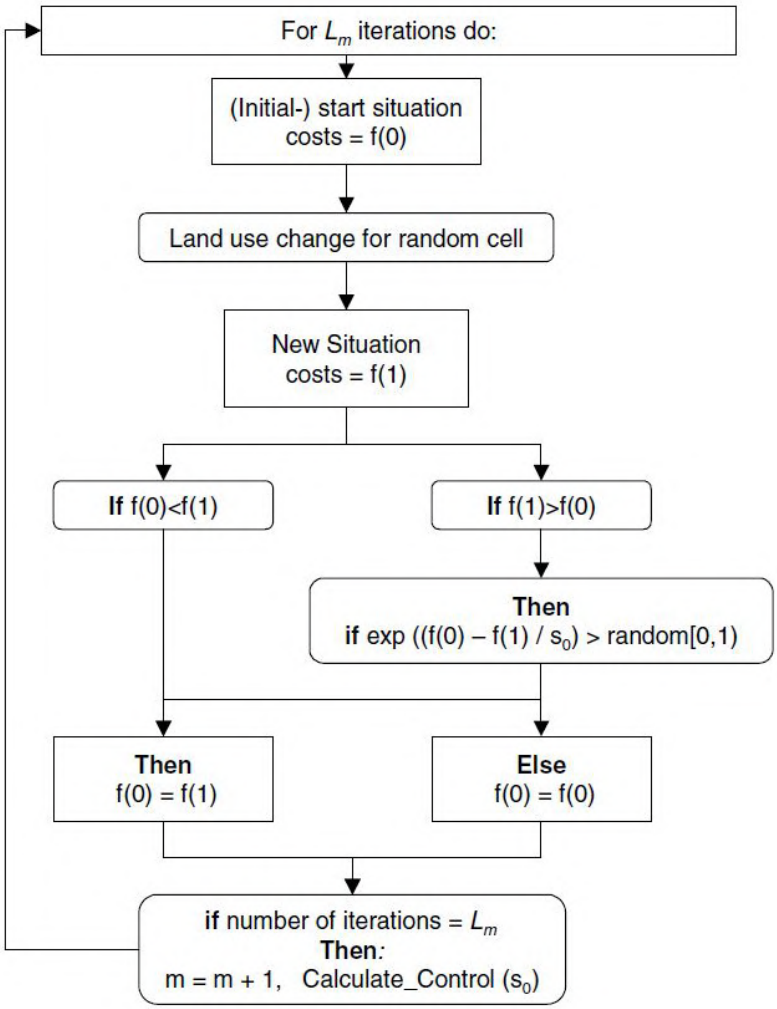


FIGURE 4.1: Flow Diagram of a Simulated Annealing Algorithm for MOLA (Aerts, Herwijnen, and Stewart, 2003)

When approaching the MOLA problem a priori, simulated annealing seems to be a feasible approach. In (Aerts and Heuvelink, 2002), it was applied to a case study with 50x50, 250x250 and 300x300 with two objectives; minimization of the costs and maximisation of the compactness. As excepted, the running time rises fast as the grid size expands. Still, the larger grids

could be optimized in hours on, according to the authors, 'average' PCs at that time. Therefore, this approach clearly outperforms the previously discussed linear programming results of (Aerts, Herwijnen, and Stewart, 2003). However, according to (Santé-Riveira, Boullón-Magán, Crecente-Maseda, et al., 2008) this approach still has one big downside; it relies heavily on starting with a relative good land-use allocation. Starting from scratch would result a high computational burden, drastically increasing the time to solve problems. Also, the approach of (Aerts and Heuvelink, 2002) was still limited to finding a single solution. In order to approximate the Pareto front, it would therefore be necessary to re-run the algorithm multiple times given different goals, which is a big disadvantage.

4.2.2 Pareto Simulated Annealing

The approach of (Aerts, Herwijnen, and Stewart, 2003) was limited to finding a single solution given one aggregated objective function. To tackle this issue, (Duh and Brown, 2007) applied a Pareto simulated annealing (PSA) approach, capable of approximating (a subset of) the Pareto front in one run.

Pareto simulated annealing was first introduced by (Czyżżak and Jaszkiewicz, 1998). It is a modified version of simulated annealing that uses a set S of interacting solutions at each iteration to propagate new solutions (Duh and Brown, 2007). This starting set is (usually) generated at random. The following sets of solutions are created by applying local search operations on previous solutions. In case a solution y is non-dominated by the solution x it originates from, it can be added to a set M in case it is not dominated by solutions of this set either. If added, the solutions of M that that are dominated by the added solution are removed. As so, set M continuously contains all non-dominated solutions found so far. If y is rejected (dominated), it can still be added to M with a chance P equal to:

$$P(x, y, T, \Lambda^x) = \min\left\{1, \exp\left(\sum_{j=1}^{D} \lambda_j^x (f_j(x) - f_j(y))/T\right)\right\} \tag{4.8}$$

hereby, $f_j(x) - f_j(y)$ equal the difference in objective value j for solution x and y, with D objectives, temperature T and weighted vector $\Lambda^x = [\lambda_1^x, \lambda_2^x, ..., \lambda_D^x]$ of the previous iteration of x. The weighted vector Λ^x aims to create dispersion among the different solutions (Duh and Brown, 2007). For each solution $x \in S$, the weights are updated after each iteration, and set to rise the chance of going away from their nearest neighbour x'. The outline of the complete algorithm is shown in Algorithm 1.

Algorithm 1 Pseudo-Code PSA (Duh and Brown, 2007)

input: a initial set of solutions S
for all $x \in S$ **do**
    update set M with x
end for
set current temperature T to initial temperature T_0
while stop conditions are not fulfilled **do**
    for all $x \in S$ **do**
        construct a solution y
        if y is not dominated by x **then**
            update set M with y
        end if
        select $x' \in S$ nearest (non-dominated) to x
        if x' does not exist (or in the first iteration) **then**
            assign weights at random, assuring $\forall j \lambda_j^{x'} \geq 0$ and $\sum_j \lambda_j^{x'} = 1$
        else
            for all objectives f_j **do**
                $$\lambda_j^{x'} = \begin{cases} \alpha \lambda_j^x & \text{if } f_j(x) \geq f_j(x') \\ \lambda_j^x / \alpha & \text{if } f_j(x) < f_j(x') \end{cases}$$
            end for
            normalize the weights such that $\sum_j \lambda_j^{x'} = 1$
        end if
        update x with y, given $P(x, y, T, \Lambda^{x'})$
    end for
    if the criteria of adjusting the temperature are met **then**
        lower T using $T(k)$
    end if
end while
output: the set M

In the research of (Duh and Brown, 2007), different combinations of the compactness maximization and cost minimization objectives are handled. In order to do so in a more efficient manner, the authors generate new solutions using two knowledge-informed rules; the compactness rule and the contiguity rule. In contrast to randomly swapping two cells, the compactness rule preferentially moves a randomly selected cell to a location that promotes compactness. For example, this could be a location that has the highest number of neighboring cells with the same cover type. On the other hand, the contiguity rule encourages contiguous pattern characteristics by moving a randomly selected cell to a location that promotes patch connectivity. For example, a location that has one or more neighboring cells of the same cover type. Both rules are proven to be more efficient and effective in maximizing compactness than conventional simulated annealing approaches for single-objective spatial pattern optimization problems.

The Pareto simulated annealing algorithm was applied to several benchmark problems with and without the knowledge-informed (KI) rules (Duh and Brown, 2007). The compactness related rule(s) showed to be more effective in approximating the Pareto front when the compactness objective was involved. However, the KI rules turned out to be less efficient in creating a diverse set of solutions. In most cases, the solutions were more diverse when

the rules where not used. This was caused by the fact that rules (for example the Compactness rule) can force PSA to preferentially explore and exploit the objective(s) that have been made easier and result in neglecting other objectives. It therefore tends to steer the search towards these objectives. As such, the rules should not be to 'aggressive' and have to much influence on the overall course of the search. The results were not compared to single-objective simulated annealing approaches. Therefore, no comments can be made on whether (multiple) a priori runs or one posteriori would be preferred in terms of time and solution quality.

4.2.3 Tabu Search

Another search heuristic applied to the MOLA problem, is tabu search (TS). Tabu search incorporates a different way of hindering the algorithm to get stuck in a local optimum or minimum. In contrast to simulated annealing, tabu search always accepts a new solution, even if it is worse then the current one (Matthews, Craw, Elder, et al., 2000). It uses a memory to prevent the search from going to places it has already been and to keep it away from local optima or minima (Sharma, 2005). All previous solutions that do not satisfy certain conditions, are regarded as 'forbidden' or 'Taboo'. In addition to the tabu-list, a long-term memory list can also be used to guide the search with a diversification strategy. This memory could direct the search to unvisited regions of the search space, by the longterm prohibition of solutions previously found.

Similarly to simulated annealing, tabu search is originally meant for single objective problem optimization. In the research of (Sharma, 2005), the MOLA problem was changed to a single-objective problem using an aggregated sum method. A flow chart of their tabu search algorithm for MOLA is shown in Figure 4.2. At first, a feasible initial solution is created. Next, a new solution was generated by randomly selecting one single land unit to swap, and several (4 or 8) potential others to swap this cell with. The best one out of these potential other cells was chosen and swapped. In case this solution improved over the objective values of current one, it is accepted. Otherwise, the solution is accepted with a chance equal to the 'potential hot-swaps' chance, which is called a hot swap. The potential hot-swap chance is calculated as follows;

$$\text{Potential Hot-Swaps} = \text{int}\left[\frac{1}{\text{Swapping Rate/No. Of Steps}}\right] \tag{4.9}$$

where the swapping rate denotes the total number of land-use swappings in a step. The higher the total number of steps taken (iterations of the algorithm), the less hot swaps will be performed. The chance will become 0 in when a certain percentage (for instance 5%) of the number of swaps for each step is reached. In case the swapping rate is met, the algorithm continues to the next step. After each step, the search strategy, neighbourhood size, tabu list and tabu length, and swapping rate can be adjusted. The tabu list contains all solutions that have been forbidden for a certain number of iterations. The tabu length states the number of iterations for which each of these solutions are considered as forbidden. (Sharma, 2005). Finally, the algorithm ends when a certain criterion is met, such as when no improvement is found after a step.

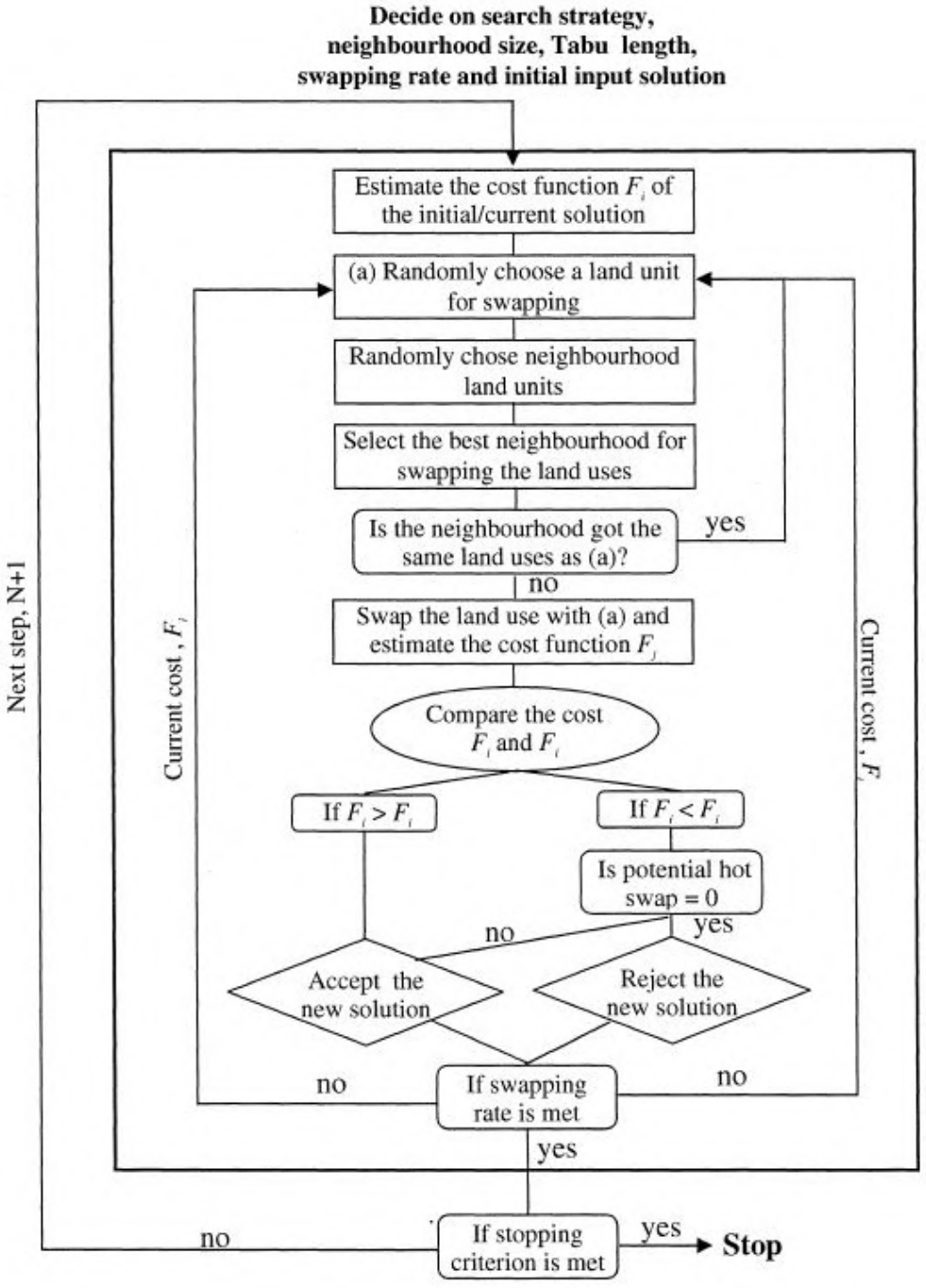


FIGURE 4.2: Flow Diagram of a Tabu Search Algorithm for MOLA (Sharma, 2005)

The advantage of a heuristic search algorithm as TS lies in the structured control of the search process. In comparison to simulated annealing, the user is simply able to have more control on its search strategy. However, is obtained at the expense of a significant book-keeping overhead (Matthews, Craw, Elder, et al., 2000). Especially storing of the tabu list could become quite expensive at higher tabu lengths. The computational time of the TS based algorithm for MOLA depended on (among others) the tabu size, swapping rate and search strategy. According to (Sharma, 2005), within the combinatorial

methods, SA still performed better than TS. On average, SA resulted in higher quality approximations given an equal running time. However, according to (Sharma, 2005) this could be improved with more future research towards appropriate search strategy and tabu list handling.

4.3 Genetic Algorithms

4.3.1 Introduction to Genetic Algorithms

Genetic algorithms (GA) are search heuristics based upon evolution (Whitley, 1994). In essence, GAs are a crude representation of natural evolution with a population improving its genetics to best survive an environment (Matthews, Craw, Elder, et al., 2000). A GA encodes solutions to a certain problem on a chromosome-like structure, and uses recombination and selection on a population of solutions to slowly improve the overall quality. In this chapter, we will discuss the basics of genetic algorithms, and how these can be used to optimize combinatorial problems.

The general flow of a genetic algorithm is shown in Figure 4.3 An implementation of a genetic algorithm begins with a population of (random) solutions (Whitley, 1994). Next, these solutions are evaluated, and their quality (fitness) regarding the problem is being rated. In case of MOLA, the fitness could depend on whether a solution dominates other solutions and/or is being dominated by others considering the previously discussed objectives. Solutions with a higher fitness, should have a higher chance of being selected as 'parents' for the creating new solutions for the next generation. These new solutions are created by crossover and random mutation operators. A crossover operator combines genes of different parents to create new off springs. In MOLA, this could for instance mean that the land-use allocations of different regions are being swapped/combined. A mutation operator randomly adjusts genes to alter these offspring, by for example swapping or adjusting a few patches of the solution. The cycle of selecting parents of the current population, and generating off springs to create the next population, continues until a certain stop criterion is met.

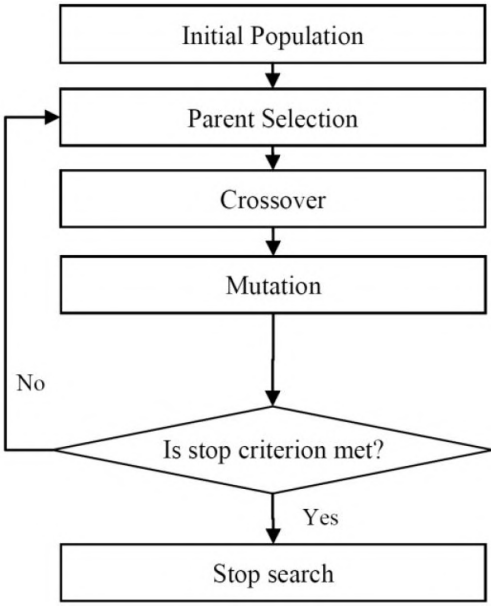


FIGURE 4.3: Flowchart of Genetic Algorithm (Heydari and Yousefli, 2017)

GAs have proven to be consistently effective in different fields of application (Matthews, Craw, Elder, et al., 2000). The robustness of GA performance can be seen in their consistent ability to find good solutions, particularly in very large and complex search spaces. Furthermore, the population based nature of their search reduces the likelihood to become trapped in local optima. When globally optimal solutions are desired, a common approach has been to combine the GA with local search heuristics such as hill-climbing to find the peak in a certain region. Genetic algorithms are a common approach for MOLA. One of the reasons for this, is the large search space of the problem. However, efficiency of a GA depends on the existence of an appropriately responsive model to evaluate solutions, an effective representation of the search space, and a compatible set of operators. The challenge of applying a GA to MOLA, is therefore to find a suitable representation and operators, that allow for an efficient convergence towards the optimum. For MOLA, a fast convergence is desirable as it allows larger scale planning problem to be analyzed interactively using planning tools.

4.3.2 Genetic Algorithms and MOLA

GAs can be applied to MOLA in two manners; using either an a priori or a posteriori approach. A priori approaches have been discussed in Chapter 3.4.1. In most cases, the goal programming approach was used as was shown in Equation (3.27). Often, the constraints of the MOLA problem, such as the bounds on the number of cells per land-use type and minimum cluster, are also added to the goal programming function. This can for instance be done in the same manner shown in Equation (3.28), (3.29) and (3.30). In Chapter 4.3.8, we will outline several of these a priori approaches and compare these to other techniques. In the second case, when a posteriori approach is chosen, the genetic algorithm discussed in Chapter 4.3.1 is modified to return a complete Pareto front. This means, that the fitness of a solutions now depends on its fitness per objective in regard to other solutions. In general, the final population of such an algorithm is the approximation of the front. An often used multi-objective genetic algorithm is the NSGA-II algorithm, which stands for

non-dominated sorting genetic algorithm. These algorithms, including their application to MOLA, will be discussed in more detail in Chapter 4.3.8 and 4.3.9.

In general, a genetic algorithm as shown in Figure 4.3, exists out of five core steps; deciding upon a chromosome representation and the four main steps of the algorithm. Different chromosome representation can be chosen, as was shown in Chapter 3.5. However, as we will see in Chapter 4.3.3, representations among previous research deviate relatively little. The first step of the algorithm, is the initialization of the first population. This can either be done randomly, or using techniques to enhance the quality of this population. The second step, is a selection of the parents that will be used for recombination. This should in some manner depend on the fitness of the solutions, where a higher fitness should increase the chance of a solution being picked. The third step is crossover, where parents exchange their genes and create new children. The crossover operators can be divided in problem dependent and independent operators, based upon whether they are specifically targeted at MOLA. The final step is mutation, that in some manner should cause for random changes among the children. In the following subchapters, we will outline how these steps have been implemented for GAs for MOLA in previous research. As the representations chosen among almost all researches deviate only little, the different techniques (such as the operators) can easily be exchanges among distinct GAs for MOLA. Except for the selection part, most outlined techniques can also be exchanged between single- and multi-objective GAs.

4.3.3 Step 1: Representation

As shown in Chapter 3.5, there are multiple possible representations of the problem. Genetic algorithms ask for a 'chromosome' representation that encodes the land-use allocations (Cao, Batty, Huang, et al., 2011). The manner in which a chromosome and its genes are encoded, affects the possible operations and their efficiency. In general, there are two categories of representations; grid-based and vector-based representations.

The vector-based representation can allow for a relatively small decision variables size, as was discussed previously in Chapter 3.5.3. As such, if implemented well, the amount of memory required to run the algorithm can be kept rather small. Still, the vector-based representation been applied in only few approaches, including the one of (Cao and Ye, 2013). Reason for this, is that the vector-based representation method also brings some challenges. First of all, the spatial relationships are much more sophisticated. It does not take note of all spatial information or relations (that might for instance be necessary for certain objectives), and could therefore require the use of a supplementary data structure (Schwaab, Deb, Goodman, et al., 2018). Secondly, it requires the user(s) to have knowledge about the area being optimized in advance (Cao, Batty, Huang, et al., 2011). These complications led to a big majority of the researchers applying a grid-based representation. Since the big majority of the genetic algorithms make use of the grid-based representation, we will continue to use this representation in the following steps as well. In case that a components is aimed at a vector-based representation, this will be mentioned and (if possible) translated to the grid-based representation as well.

Finally, as for as concerned, no research has implemented the quad-tree representation or the patch vector representation of Chapter 3.5.2 and 3.5.4. One approach, by (Datta, Deb, Fonseca, et al., 2007), makes use of the the three-dimensional model as formulated in Chapter 3.5.5. In this model, the grid-based representation is extended with a third-dimension to represent time.

4.3.4 Step 2: Initialization

Theoretically, evolutionary algorithms are not dependant on the the initial population. However, the quality of the initial population can have a significant influence on the convergence speed of the GA (Datta, Deb, Fonseca, et al., 2007). These initial solutions do not necessarily have to be feasible, in case the GA has access to operators to fix infeasible solutions (Chapter 4.3.7). According to (Datta, Deb, Fonseca, et al., 2007), it is observed that guidance in the initialization for creating (nearly) feasible solutions, can help in a drastic reduction of computational time for the GA. However, the initialization is also of importance to the quality of the GA's convergence (Cao, Huang, Wang, et al., 2012). For example, one has to be careful not to use any settings for initialization that may lead to local optima only. In the this section, we will outline some initialization approaches used in previous research in more detail.

The most simple initialization approach, is by simply creating random solutions only. This method is, among other, applied in (Cao, Huang, Wang, et al., 2012) and (Cao and Ye, 2013). Advantage of this approach, is its simplicity and (probably) high divergence in solutions, decreasing the chance of ending up in local minima. However, this approach could require a lot of computational effort to converge. Another approach, is to generate part of the initial population at random, and fill up the other part with solutions that represent the current situation or 'status quo' of the area. This initialization operator is named the problem-based initialization operator (PBIO). In (Cao, Batty, Huang, et al., 2011), 90% was created at random and 10% was status quo. In this manner, more information regarding the current situation remains involved, which can for example be relevant for maintaining low development costs. Another simple approach that tends to retain part of this information, was proposed in (Song and Chen, 2018). In their approach, they randomly mutated part of the current situation (30%), while keeping the rest intact (70%). A balance has to found between between the amount of randomly created solutions and initial ones that support an efficient convergence.

Knowledge-informed initialization strategies tend to create initial solutions using properties for MOLA. The solutions hereby already respond to the objectives for as far as possible (Duh and Brown, 2007). One common knowledge-informed initialization strategy based upon the use of a selection value, was proposed and used in (Stewart, Janssen, and Herwijnen, 2004) and (Aerts, Van Herwijnen, Janssen, et al., 2005). The strategy worked as follows. Every solution was generation by first randomly selecting one cell (i, j), to which a random type k was given. The chance of assigning a certain type to this cell equals σ_{ijk} (the selection value). The selection value indicates the benefit of assigning land-use k to cell (i, j). Hereby, $d_{ijk} = 0$ when constraints prohibit allocation of land-use k to this cell. The higher σ_{ijk}, the higher the chance that

type k will be allocated to (i, j). In (Stewart, Janssen, and Herwijnen, 2004) the chance $P_{ij}(k)$ for selecting land-use x of $x = 1, ..., K$ for cell (i, j) is calculated linearly as follows;

$$P_{ij}(x) = \frac{d_{ijx}}{\sum_{k=1}^{K} d_{ijk}} \tag{4.10}$$

After the randomly selected first cell has received a type, a random neighbor of this cell is selected. Again, a land-use type is chosen based upon d_{ijk} for this new cell (i, j). This process continues until a land-use type has been allocated to all cells. The specific calculation of d_{ijk} can for example be done by summing up factors that contribute to the preference of k. In (Stewart, Janssen, and Herwijnen, 2004), two factors are named that can be used to calculate the selection value;

(1) A factor to encourage aggregations into clusters, when compactness is being maximized. This factor should increase d_{ijk} with the number of neighbouring cells of (i, j) that also have land-use type k. In (Stewart, Janssen, and Herwijnen, 2004), this factor is defined as θ^{v_k}, where v_k is the number of neighbouring cells already allocated to land-use k, and θ is the 'tuning factor'.

(2) A factor to encourage achievement of the target numbers for each land-use, when these are bounded by constraints. In (Stewart, Janssen, and Herwijnen, 2004) this is done by first randomly selecting nominal target values S_k for each land-use k within the given bounds. Next, the factor can be defined as $\frac{S_k - S_k^c}{S_k}$, where S_k^1 is the number of cells allocated to land-use k up to this stage.

As mentioned, the process continues until a land-use type has been allocated to all cells. In case no unallocated neighbour cells can be found before this is met, another unallocated cell is selected at random and allocated. It can occur that the area constrains certain cells to be converted to certain land-use types. This can either be problem-specific or due to the choice not to alter a certain part or percentage of the current situation. In that case, the selection value $P_{ij}(x)$ can than be defined as follows (Li and Parrott, 2016);

$$P_{ij}(x) = B_{ijx} \times \frac{d_{ijx}}{\sum_{k=1}^{K} d_{ijk}} \tag{4.11}$$

where $B_ij(x)$ is 0 if cell (i, j) may not be converted to land-use type k, and 1 otherwise. The advantage of this knowledge-informed strategy is the high level of control that the land-use planner has due to the selection value function. Disadvantage, is the fact that it costs more computational time to generate an initial population than when generated randomly. Also, it may not be desirable to make the selection function very 'extreme' or specific in a manner that it would significantly decrease variation between the initial solutions.

4.3.5 Step 3: Selection

Selection is the procedure in which individuals are selected for reproduction (Cao and Ye, 2013). Hereby, the selection operators should promote fit solutions and nullify the less fit ones. This simulates the natural selection process, where genes with a higher fitness have a higher chance to reproduce, resulting in an overall increase in fitness throughout the populations. As such, the selection operator should provide higher probability of choosing a solution for a crossover with a high fitness value (Aerts, Van Herwijnen, Janssen, et al., 2005). Different selection mechanisms can be chosen, which will be outlined in the next paragraphs. In contrast to other GA components, most selection techniques for single-objective GAs are not efficient (or even possible) for multi-objective GAs. As such, we will handle those separately. At first, we will be discussing methods for single-objective GAs for MOLA, of which fitness proportionate selection is most common. Finally, we will discuss selection methods for multi-objective GAs for MOLA, of which the NSGA-II selection mechanism is applied most often.

Selection In Single-Objective GAs

A commonly used selection operator in single-objective genetic algorithms, is fitness proportionate selection, also called roulette wheel selection. In fitness proportionate selection, the probability of a particular solution being selected as a parent is a function of a single fitness value function (Stewart, Janssen, and Herwijnen, 2004). As for a multi-objective problem such as MOLA, this means that the objectives have to combined with for instance an a priori method of Chapter 3.4.1. Next, a solution is chosen with relative probability based on this function's value. A simple and common approach to do so, based upon linear interpolation, works as follows (Aerts, Van Herwijnen, Janssen, et al., 2005); First, calculate the fitness of all solutions using an a priori technique such as goal programming. A (relative) probability of 1 is given to the individual with the most optimal objective value and a probability of ζ is given to the individual with the worst. The value of ζ needs to be between 0 and 1. Next, the probabilities of the other solutions are calculated by linearly interpolating. Next a solution will be chosen, where the chance $P(x)$ of a solution $x \in S$ with interpolated value x_p ($\zeta \leq x_p \leq 1$) is equal to;

$$P(x) = \frac{x_p}{\sum_{y \in S} y_p} \tag{4.12}$$

To allow for additional tuning of the algorithm, the value of ζ is allowed to vary during the process (Aerts, Van Herwijnen, Janssen, et al., 2005). This can for example be done by defining two values ζ_0 and ζ_1, such that ζ is equal to ζ_0 in the first generation and linearly increases until its value is ζ_1 in the last (predefined) generation. Finally, the selection method can be combined with other methods such as elitism selection, as was done in (Cao and Ye, 2013). With elitism selection, the best k number or % of solutions are always copied directly to the next generation. This can be used to overcome the (small) chance of the individuals with the best fitness values not getting selected for the next generation.

Selection In Multi-Objective GAs

In multi-objective genetic algorithms, it would not be suitable to base the selection chance upon an a priori fitness function, as we are searching for the complete Pareto front. As such, the selection chance in multi-objective GAs is often based upon a solution's dominance by other solutions of the population. The 'less' a solution is dominated by other solutions, the higher its chance should be to be selected for the next generation.

The most adopted MOGA is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) algorithm. This algorithm provides a mechanism for selecting candidates for the next population and deciding upon the fitness of its solutions. As NSGA-II is the most applied multi-objective GA for MOLA, we will now discuss its selection procedure in more detail. The overall outline of the selection procedure of the algorithm is visualised in Figure 4.4.

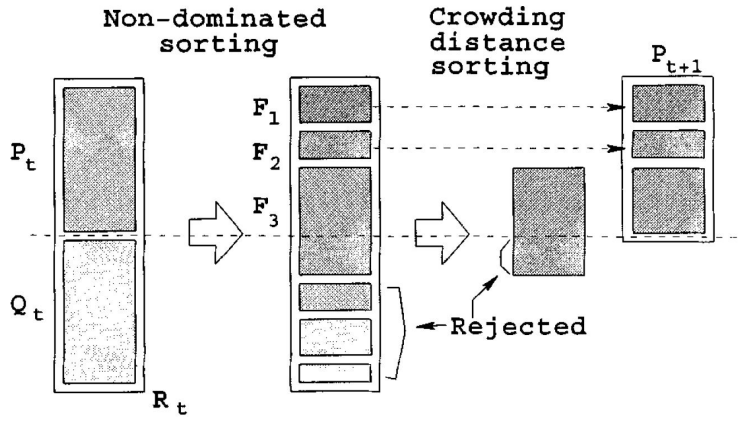


FIGURE 4.4: Selection in NSGA-II(Deb, Pratap, Agarwal, et al., 2002)

The algorithm starts off with population P_t, that creates an offspring population Q_t of equal size. The parent selection mechanism for recombination will be discussed later. First, we will look at how the next population P_{t+1} is selected out of P_t and Q_t. In order to do so, we need to determine which solutions in this multi-objective space are considered 'best'. This is done using non-dominated sorting and the crowding distance property.

Algorithm 2 Pseudo-Code Fast-Non-Dominated-Sort (P) (Deb, Pratap, Agarwal, et al., 2002)

```
1:  for all p ∈ P do
2:      S_p = ∅
3:      n_p = 0
4:      for all q ∈ P do
5:          if p ≺ q then
6:              S_p = S_p ∪ {q}
7:          else
8:              if q ≺ p then
9:                  n_p = n_p + 1
10:             end if
11:         end if
12:     end for
13:     if n_p = 0 then
14:         p_rank = 1
15:         F_1 = F_1 ∪ {p}
16:     end if
17: end for
18: i = 1
19: while F_i ≠ ∅ do
20:     Q = ∅
21:     for all p ∈ F_i do
22:         for all q ∈ S_p do
23:             n_q = n_q − 1
24:             if n_q = 0 then
25:                 q_rank = i + 1
26:                 Q = Q ∪ {q}
27:             end if
28:         end for
29:     end for
30:     i = i + 1
31:     F_i = Q
32: end while
```

At first, the complete population P_t and Q_t are sorted based upon their dominance. As discussed in Chapter 2, a solution is said to be dominated by another solution, if the other solution can improve on at least one of its objective values without degrading any of the other objective values. The non-dominated sorting procedure sorts the solutions in 'ranks'. Hereby, solutions in F_1 are dominated by no other solutions, F_2 are dominated by solutions from F_1, F_3 by solutions from F_2 and F_1, and so on. The procedure of this sorting is shown in Algorithm 2 (Deb, Pratap, Agarwal, et al., 2002). With M objectives and population size of N, it has a complexity of $O(M(2N)^2)$ (Zitzler, Deb, Thiele, et al., 2001). The lower the rank, the 'better' we consider this solution to be. Hence, the solutions are ordered by rank, and the ones with the lowest ranking are taken to the next population.

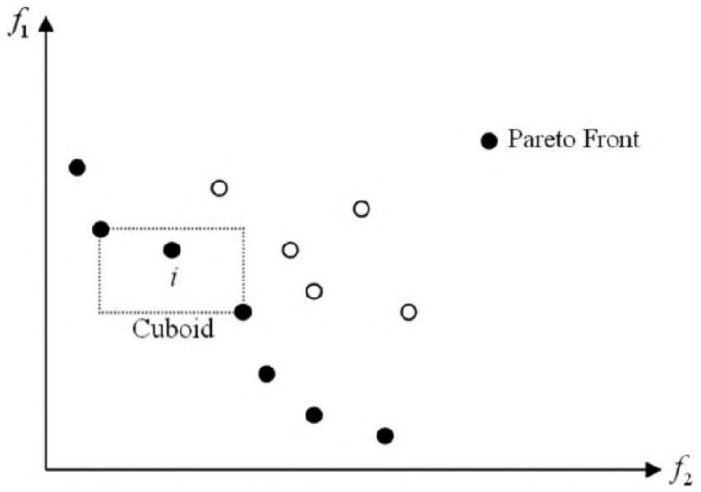


FIGURE 4.5: Crowding Distance of a Solution (Cao, Batty, Huang, et al., 2011)

Still, it can occur that we are not able to take all solutions of a certain ranking to the next population. As such, we also need to be able to decide upon the quality between solutions in the same non-dominated sorting rank. Therefore, the crowding distance property is used. The crowding distance gives an indication of the solution density in its neighbourhood, as shown in Figure 4.5. For each objective, all solutions are sorted and the distance between a solution's 'previous' and 'next' neighbour is added to its crowding distance. The exact calculation of the crowding distance is shown in Algorithm 3. With M objectives and population size N, it has a complexity $O(M(2N)\log(2N))$ (Zitzler, Deb, Thiele, et al., 2001). The higher the crowding distance, the lower the density of solutions in its neighbourhood. In general, these solutions are preferred above solutions in dense areas, as this promotes 'spread' among the solutions (and genes) in the population. As such, the solutions with the lowest crowding distance are taken to the next population.

Algorithm 3 Pseudo-Code Crowding-Distance-Assignment ($\mathcal{I}$) (Deb, Pratap, Agarwal, et al., 2002)

$l = |\mathcal{I}|$
for all solution $i \in \mathcal{I}$ **do**
    set $\mathcal{I}[i]_{distance} = 0$
end for
for all objective $m \in \mathcal{M}$ **do**
    $\mathcal{I} = sort(\mathcal{I}, m)$
    $\mathcal{I}[1]_{distance} = \mathcal{I}[l]_{distance} = \infty$
    for $i = 2$ to $(l-1)$ **do**
        $\mathcal{I}[i]_{distance} = \mathcal{I}[i]_{distance} + \frac{\mathcal{I}[i+1].m - \mathcal{I}[i-1].m}{f_m^{max} - f_m^{min}}$
    end for
end for

Using non-dominated sorting and the crowding distance, we have been able to decide upon the solutions for the next population. Finally, we need to decide upon what parents are going to produce offerings for the next population. In NSGA-II, this is usually done by doing binary tournaments (k = 2) with randomly selected solutions from population P. The one with

the lowest non-dominated sorting rank wins, or in case the rank is equal, the one with the highest crowding distance. Sorting all solutions on fitness costs $O(2N\log(2N))$, with population size N. Note, that the non-dominated sorting procedure has the highest complexity, and so dominates the overall NSGA-II's complexity.

Although most researches apply the standard NSGA-II selection mechanism to MOLA, some variations on it have been introduced and tested on MOLA. In (Song and Chen, 2018), a variation on NSGA-II is used where parents are selected for reproduction at random. According to the researchers, random parent selection has proven to work best with two-point-crossover (Chapter 4.3.6) and swap-mutation (Chapter 4.3.7). Nevertheless, the general conviction is that parent selection based upon dominance and/or crowding-distance leads to a faster convergence. As such, the vast majority of multi-objective genetic algorithms for MOLA implement this standard selection method.

4.3.6 Step 4: Crossover

In order to recombine solutions, crossover (also named recombination) is performed, which is the process of exchanging genes between solutions (parents) of the current population, to create new solutions (offsprings) (Cao, Huang, Wang, et al., 2012). In order for a GA to realize an effective optimization process, building blocks need to be formed and passed on through consecutive populations in a manner that allows fit genes to survive. As such, suitable crossover operators need to be chosen, that avoid the destruction of these building blocks. In this section, several crossover operators will be outlined, divided in four different categories. The four categories are defined using two (independent) operator properties. First of all, a crossover operator can either be problem dependent or problem independent (Cao, Huang, Wang, et al., 2012). Problem independent operators do not exploit any characteristics of the MOLA problem, where problem dependent operators do. In the case of MOLA, a problem dependent operator could for example focus on maintaining or increasing the compactness of its offsprings. Secondly, crossover operators can be categorized using the chromosome representations on which they operate. In the case of MOLA, previous research mentions operators for the grid- and vector-based representation. Organized in four categories using these two properties, the following paragraphs will outline the most common crossover operators for MOLA.

Finally, note that after applying most of the crossover operators, the feasibility of an offspring solution can not be always be guaranteed. It could for example occur that offsprings violate the allocation ranges constraint of Equation (3.3). In that case, a reparation mutation may need to be applied, in order to fix this solution (Datta, Deb, Fonseca, et al., 2007). Reparation mutations will be discussed in more detail later in Chapter 4.3.7.

Problem Independent Grid Crossover

A problem independent operator can be applied to any problem, whereas a problem dependent operator is specifically designed for one problem, such as MOLA. As such, exchanging a square or circle of a grid would be problem

independent, where the exchange of a land-use cluster would be problem dependent. The advantage of a problem independent operator, is that they are generally less complicated and computationally expensive. The downside, is that they are often less efficient in optimizing the problem objectives then tailored problem dependent operator.

The first independent crossover operator for grid-based MOLA that will be discussed, is the two-dimensional crossover operator (TDX). The TDX operator works as follows. First, the grid is split up in four squares, by selecting a row and a column at random. Next, these four blocks are recombined to create new offsprings. This can be done in different manners. In (Datta, Deb, Fonseca, et al., 2007), one random block was chosen and swapped. In (Li and Parrott, 2016), the first (left-top) and fourth (right-bottom) are always swapped. The latter is depicted in Figure 4.6.

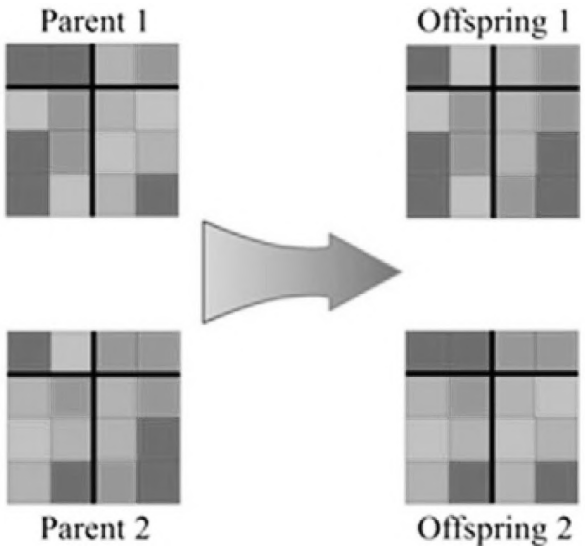


FIGURE 4.6: The TDX Operator (Li and Parrott, 2016)

In the work of (Schwaab, Deb, Goodman, et al., 2018), multiple spatial crossover operators were applied to MOLA, of which the results were compared. The tested operators are visualised in Figure 4.7. The shown grids are the offsprings after using the associated operator, where the darker areas contain land-uses of one parent, and the light areas contain land-uses of the other. The operators work as follows;

1. Uniform Crossover (UC): In this type of crossover, information is (typically) randomly exchanged between individual cells of the grid. In some cases, other mixed ratios are used. Note, that this crossover type is non-spatial, as it could also be applied to a vector.

2. Vertical/Horizontal Crossover (VC/HC): One random column (VC) or one random row (HC) is selection, after which the parents are split into two and recombined to new offsprings. The previously discussed TDX operator is a combination of both.

3. Vertical/Horizontal Band Crossover (VBC/HBC): Two random columns (VBC) or random rows (HBC) are selection. The cells inside the created region (band) is exchanged between the parents.

4. Angle Crossover (AC): A line is drawn that intersects the middle of the grid with a random angle. Next, the parents are split and recombined across this line.

5. Block Crossover (BC): Two random columns and two random rows are selected. The information inside the block formed between these four lines is exchanges between parents.

6. Block Uniform Crossover (BUC): Blocks are formed according to a certain predefined shape, such as pairs of two as in Figure 4.7. Next, uniform crossover is applied on these blocks.

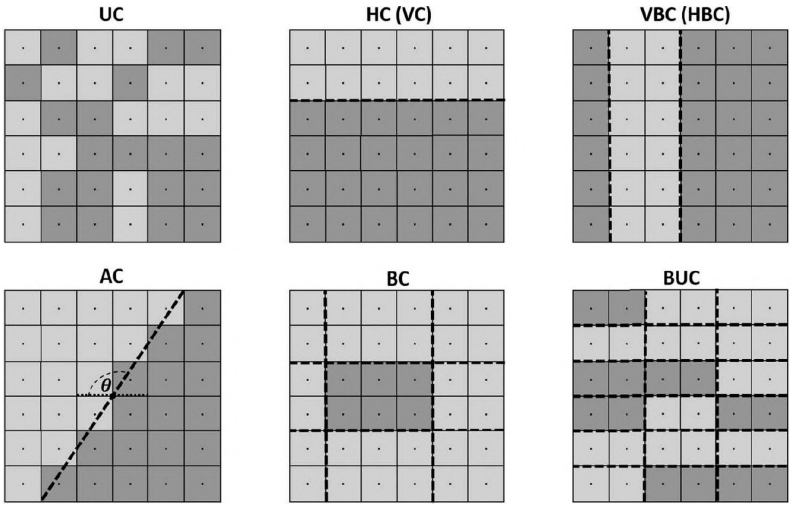


FIGURE 4.7: Six Problem Independent Spatial Crossover Operators (Schwaab, Deb, Goodman, et al., 2018)

The performance of the previous six operators was tested on MOLA in combination with different mutation operators in (Schwaab, Deb, Goodman, et al., 2018). However, results show that, when only using one operator in combination with a basic NSGA-II algorithm, there are no significant differences in their performance. The VBC and HC seem to perform slightly better, but again these differences are of such small magnitude that they can be neglected. The authors do not provide any explanation or justification of the results obtained. Eventually, the crossover operators were combined with mutation operators, to obtain more insight in what operators perform best. This will further be discussed further in Chapter 4.3.7.

Problem Dependent Grid Crossover

Since the previously independent operators do not take any objectives and/or problem properties in account, this can lead to an inefficient optimization process. In fact, they can be very disruptive for clusters, which can be undesirable when optimizing compactness. Therefore, problem dependent crossover operators are, despite of their higher complexity, often preferred. By taking the objectives of MOLA in account and understanding how/when these are optimized, operators can be formed to stimulate these objective values in the

resulting offsprings.

The problem dependent crossover operators for MOLA with a grid representation all have one aspect in common; they focus on the boundary cells (or edge nodes) of a solution. A boundary cell is a cell that has at least one neighbor cell with a different land-use type (Song and Chen, 2018). The reason for these crossover operators to target boundary cells, is due to the fact that compactness depends on the land-use type of neighbouring cells (Equation (3.13)). The compactness is minimized if a cell has as much neighbouring cells of the same type as possible, meaning that you prefer to have as less boundary cells in your solution as possible. Throughout previous researches, different terms and notations are being used for crossover operators that depend upon boundary cells. In this research, we will refer to these operators as boundary cell crossover (BCX) operators.

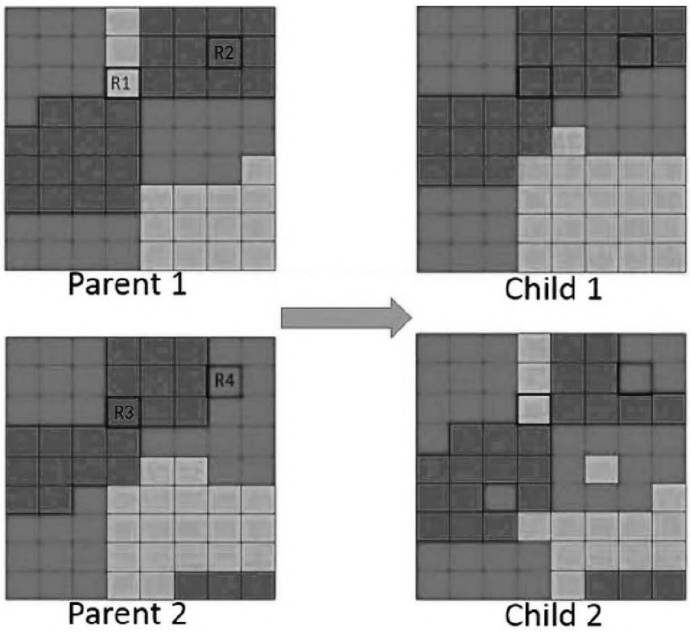


FIGURE 4.8: The BCX-II Operator (Song and Chen, 2018)

The first BCX operator was introduced by (Datta, Deb, Fonseca, et al., 2007). The operator works as follows. First, a random Hamming cell is chosen. A Hamming cell is a cell of which the land-use differs between the two parents. If the Hamming cell is on a boundary cell of the first parent, its type is swapped with the type of that cell of the second parent. This means that in case the cell only lays on a boundary of one of the two parents, only one parent will be modified. The modification was applied to all Hamming cells with a certain probability, modifying only a percentage of the Hamming cells.

A slightly modified version of BCX, which we will call BCX-II, was defined by (Song and Chen, 2018). BCX-II only looks at the boundary cells of the first parent. Meaning, that if the Hamming cell is a boundary cell of the first parent, the land-uses are exchanged between both parents, and otherwise not. The process of BCX-II is shown in Figure 4.8. All Hamming cells are checked and swapped, meaning that the probability of the modification is set to 100%. The BCX-II operator is computationally less expensive then BCX, as you only

have to check whether a Hamming cell is a boundary cell in the first parent. On the downside, clusters from the second parent can now be fragmented, as can be seen in the second child of Figure 4.8.

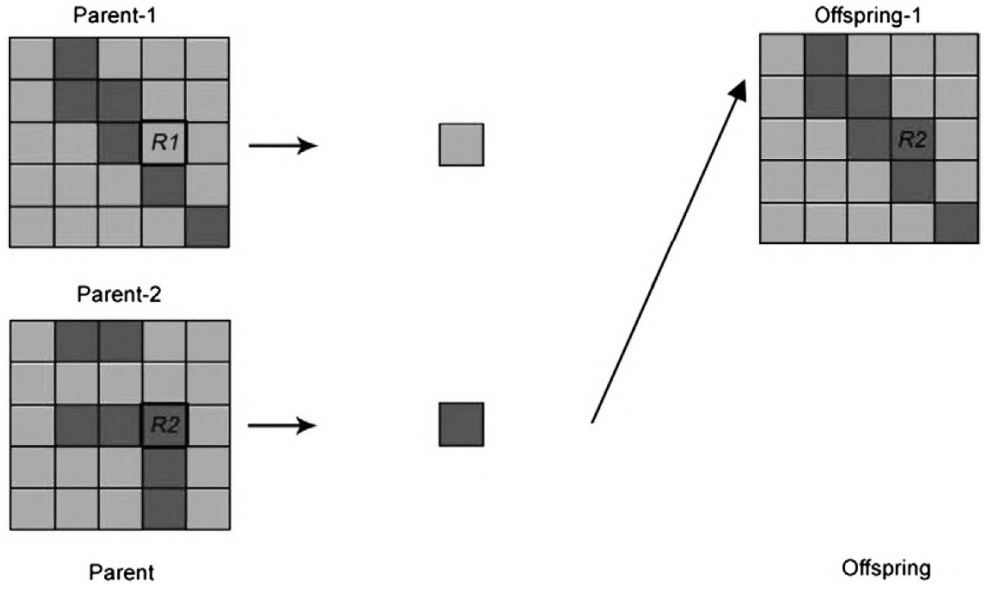


FIGURE 4.9: The BCX-III Operator (Cao, Huang, Wang, et al., 2012)

The previous crossover operators only focused on whether the Hamming cell was a boundary cell or not. In (Cao, Huang, Wang, et al., 2012), a boundary cell operator was introduced that also takes the land-use types of neighbouring cells into account. The operator, which we will refer to as BCX-III, works as follows. At first, a random Hamming cell is selected that is a boundary cell on either one of the parents. Now, the land-use type of this cell is only copied to the other parent, if afterwards it will have at least one neighbour of the same type. This means that, in both directions, the land-use types of the neighbouring cells of the Hamming cell need to be checked to assure that at least one of them is of the same type. The process is depicted for one direction in Figure 4.9. The operation will be repeated with a certain crossover rate for all Hamming cells.

A slightly modified version of BCX-III, called BCX-IV, was introduced by (Li and Parrott, 2016). It is shown in Figure 4.10. In this crossover operator, a minimum number of neighbouring cells is defined that should be of the same type as the land-use type being copied. In case of the image, this minimum number is set to two. As can be seen, this means that the cell type of parent 1 is copied to parent 2, but not the other way around. Note, that this operator can either be applied with four or eight neighbours. The advantage of taking the land-use types of neighbours in account, is that this avoids copying a land-use type that will form a new cluster of size one. Using BCX and BCX-II, this could occur, which is undesirable with respect to the compactness objective.

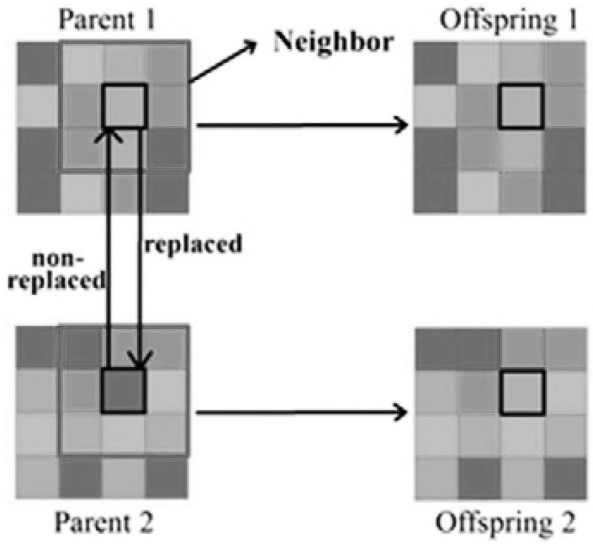


FIGURE 4.10: The BCX-IV Operator (Li and Parrott, 2016)

Problem Independent Vector Crossover

In this section, possible independent vector crossover operators for the MOLA problem will be outlined. As mentioned before, the amount of research defining and implementing vector-based crossover operators is relatively small compared to grid-based operators.

In (Cao and Ye, 2013), the single-point crossover (SPX) operator is used to recombine vectors for the MOLA problem. The reason this operator was applied, was for its simplicity. Compared to previously discussed grid-based crossover operators, the operator also has a very low computational complexity. The SPX operator works as follows. First, a crossover point is chosen somewhere on the vector, where both vectors are 'cut' in two. Next, a part of each parent is swapped, in order to form two new off springs. The process of SPX is depicted in Figure 4.11

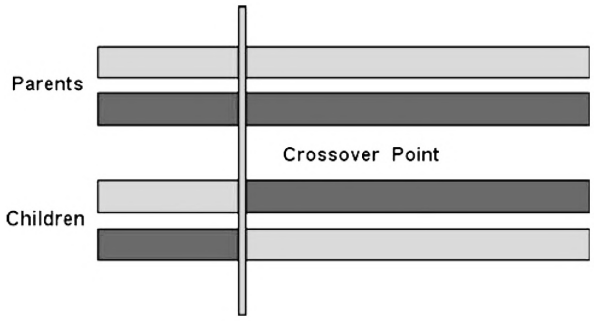


FIGURE 4.11: The SPX Operator (Cao and Ye, 2013)

Except for single-point crossover, other problem independent vector-based crossover operators could be applied tot MOLA as well. However, as far as we aware of, that has not yet been implemented in any previous research. Examples of possible independent vector crossover operators that could be used are k-point and uniform crossover. With k-point crossover, multiple (k) random points are being chosen, after which the segments that are being formed between these points are exchanged. With uniform crossover, the

land-use type of each cell will be exchanges with a certain probability. As was discussed previously, this operator can and was also applied to grid-based MOLA (Matthews, Craw, Elder, et al., 2000). Note, that these crossover operators might not be suitable when optimizing compactness, as they can easily create more fragmentation among a solution's clusters.

Problem Dependent Vector Crossover

As far as concerned, no previous research has been performed towards problem dependent crossover operators for MOLA using a vector representation. One of the reasons for this lack of research, could be the fact that it is more difficult to retrieve spatial information from a vector than a grid. When working with vectors, additive information is required for the spatial relations between the cells or regions represented in the vector. This makes it more difficult to for example identify boundaries than when using a grid-based representation.

4.3.7 Step 5: Mutation

Mutations in a genetic algorithm, are comparable with biological mutations. Mutations increase the diversity in the genes of the population, decreasing the chance of being trapped in a local minimum (Cao and Ye, 2013). By introducing changes that would not arise through crossover only, the performance of a GA can be increased significantly (Schwaab, Deb, Goodman, et al., 2018). Contrariwise, too man mutations can also negatively influence the convergence and quality of the optimization, in case too many good building blocks are destroyed.

In the following paragraphs, several mutation operators will be outlined. Similar to crossover, a mutation could lead to an infeasible solution, as the offspring might for example violate the minimum cluster size (Equation (3.16) or minimum/maximum allowed number of cells per land-use type (Equation (3.7)). In order to 'fix' this, several 'repair' mutations can be used, which will also be introduced in this chapter. All operators are once more assigned an operator code for referencing. Similar to the crossover operators of Chapter 4.3.6, a distinction will be made between operators that are either problem dependent or independent, and grid- or vector-based.

Problem Independent Grid Mutation

In this section, multiple problem independent grid mutation operators will be discussed that can be applied when using grid-structured chromosomes.

The most straightforward mutation operator, is the random cell mutation (RCM) operator. It was applied in combination with NSGA-II, in the research of (Masoumi, Maleki, Mesgari, et al., 2017). When applied to MOLA, the operator simple picks a random cell and assigned a random different land-use type to it. It can be applied to either one single cell, or to all cells with a certain mutation probability (rate).

An alternative to RCM, is the random cell swap mutation (RCSM). This operator was applied to MOLA and tested in the research of (Schwaab, Deb, Goodman, et al., 2018). The operator is, among with several others that will

be discussed later, visualised in Figure 4.12. Two random cells of different land-use types are selected, of which the land-use types are swapped. Note, that although the constraint concerning the number of cells per land-use type can not be violated, the minimum cluster size constraint can.

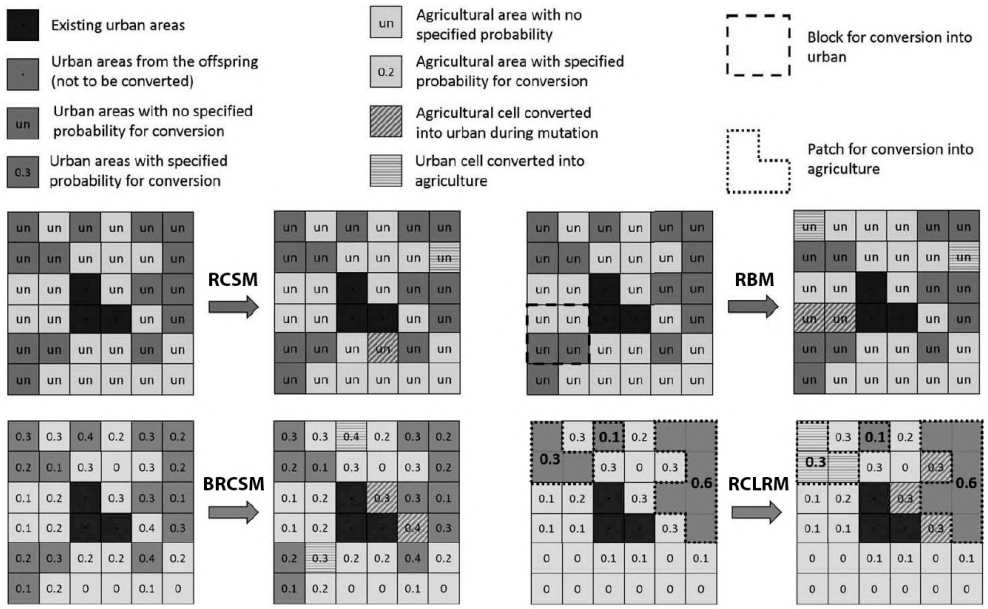


FIGURE 4.12: The RCSM, BRCSM, RBM and RCLRM Operators (Schwaab, Deb, Goodman, et al., 2018)

A more disruptive mutation operator can be created, by mutating complete blocks instead of separate cells. The most straightforward block mutation operator, is random block mutation (RBM), as was implemented in (Schwaab, Deb, Goodman, et al., 2018). A random location is chosen, where a block of cells of predefined size is completely converted to a random land-use type. The cells in this block that already had this type, remain untouched. The operator is shown in Figure 4.12.

Instead of converting a block to one random type, it is also possible to re-execute the initialization procedure on this block. This will be referred to as the random block initialisation mutation (RBIM) and was applied on MOLA in (Aerts, Van Herwijnen, Janssen, et al., 2005). The operator is knowledge independent, as the operator itself does not take the problem (initialization) in account.

Problem Dependent Grid Mutation

In this section, multiple problem dependent grid mutation operators for MOLA will be discussed that can be applied when using grid-structured chromosomes.

The problem independent RCM operator converts a random cell, which could lead to the fragmentation of existing or creation of new clusters, decreasing the compactness. Hence, biased random cell mutation (BRCM) was introduced, which takes the compactness in account. It simply increases the chance of choosing a land-use type proportionate to the number of neighbours having this type as well. In the case of (Li and Parrott, 2016), this was

implemented using the following formula;

$$MP_{ijk} = B_{ijk} \times \left(\frac{N_{ijk}}{8}\right)^2 \tag{4.13}$$

where MP_{ijk} is the (unnormalized) conversion probability of cell (i, j) to type k, N_{ijk} is the number of (the eight) neighbouring cells of cell (i, j) with type k, and B_{ijk} is 1 in case type k is allocated at cell (i, j) (otherwise 0).

Similar to with RCM and BRCM, we can also introduced a biased version of RCSM. This was implemented by (Schwaab, Deb, Goodman, et al., 2018), and will be referred to as biased random cell swap mutation (BRCSM). The operator is visualised in Figure 4.12. First, two random land-use types k are chosen. Now, for each k a cell is chosen that will be swapped, of which the chance of selecting this cell is inversely proportionate to its number of neighbouring cells with the same k. This means, that the more a cell is surrounded by cells of its own type, the less chance it has to be swapped. This decreases the chance of clusters being fragmented or new ones being formed, resulting in an overall higher compactness. In the research of (Schwaab, Deb, Goodman, et al., 2018), the four-cell neighbourhood (Von Neumann neighbourhood) is used to do so. The operator can also be used to stimulate other objectives than compactness in a similar manner. In that case, the probability of a cell conversion is biased according to the (other) objective value(s) of its neighbours.

Besides mutating cells separately, it is again possible to mutate bigger spatial forms as well. A simple problem dependent mutation to do so, is the random cluster mutation (RCLM) operator. This operator was applied to MOLA in (Li and Parrott, 2016). First, a land-use cluster composed of adjacent cells of the same type is selected. Next, all cells of this cluster are converted to the same land-use, which is randomly selected from the set of all possible land-uses. A possible extension of this operator could be to make the chance of selecting a cluster irreversibly proportionate to its size, as it can be very disruptive when applied to large cluster.

Another disruptive cluster mutation operator, was proposed by (Schwaab, Deb, Goodman, et al., 2018). The operator, which we will refer to as the random cluster removal mutation (RCLRM), removes a complete cluster and reattaches it to other clusters of equal type. The authors describe the operator for a MOLA problem with two land-use types, which is done as follows. At first, a random cluster is selected, with the probability of a cluster getting selected, being inversely proportional to its size. Next, the cluster is completely removed, meaning that it is converted to the other (second) land-use type. Finally, the same number of cells the cluster had, are 're-attached' to boundaries of other remaining clusters of the same type. The RCLRM procedure is shown in Figure 4.12. The operator can be expended to support MOLA with more than two land-use types as well, by for example converting the selected cluster to a random other land-use type.

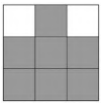


FIGURE 4.13: Example Patch Window (Cao, Huang, Wang, et al., 2012)

A more often used spatial form for mutations on MOLA, is a randomly generated patch. Random patch mutation (RPM) is the most basic form of patch mutation and was applied to MOLA in (Cao, Huang, Wang, et al., 2012). The RPM procedure works as follows. First, a patch is created by randomly selecting 7 cells from a 3x3 grid as visualised in Figure 4.13. Selecting 7 cells will always result in one continuous patch, which is beneficial for the compactness. Next, the patch is placed at a random location in the grid, and all cells inside are converted to a random land-use type. The complete RPM procedure is visualised in Figure 4.14.

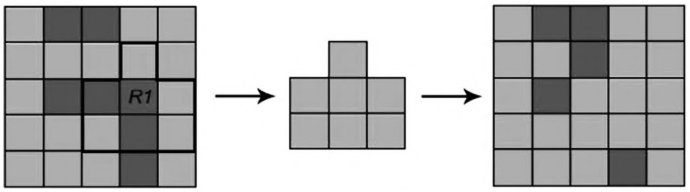


FIGURE 4.14: The RPM Operator (Cao, Huang, Wang, et al., 2012)

A second slightly modified version of RPM, which we will refer to as RPM-II, was proposed by the same authors (Cao, Huang, Wang, et al., 2012). The RPM-II operator uses the same procedure as RPM, but only converts the cells in the patch if one of its neighbouring cells has the same land-use type as the patch will retrieve. In that manner, it assured that the patch will not introduce a new cluster of this color. It is still able to fragment surrounding clusters of different land-use types.

A third variation of the RPM operator was introduced by (Cao, Batty, Huang, et al., 2011). The operator, which we will call RPM-III, sets the color of the patch to the most occurring land-use type of its current 7 cells. In this manner, the chance of cluster fragmentation is again decreased. Also, the constraint concerning the allowed number of cells to which each type may be allocated is violated less quickly. The RPM-III operator was applied to MOLA in the research of (Song and Chen, 2018).

Finally, the patch form can also be used to perform a swap, as is done with random patch swap mutation (RPSM) (Cao, Batty, Huang, et al., 2011). First, a patch is formed using the same approach as with RPM. Next, two random locations are chosen for the patches, while making sure that the patches do not overlap. Next, the land-use types of the patches were swapped. Swapping ensures that the number of cells per land-use type remains equal. The minimum cluster size can still be violated.

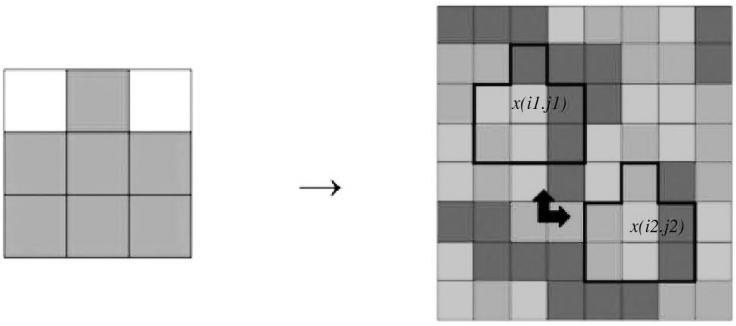


FIGURE 4.15: The RPSM Operator (Cao, Batty, Huang, et al., 2011)

Similar to most problem dependent crossover operators, several mutation operators focus on the boundary cells of a solution. Most straightforward, is the random boundary cell mutation (RBCM), as was proposed in (Datta, Deb, Fonseca, et al., 2007). The operator works as follows. First, all boundary cells are detected. Next, one boundary cell is selected at random, and its land-use type is converted to the type of one of its neighbours (chosen at random). Note, that the procedure of sorting all boundary cells can be rather expensive. The authors mention this to be necessary, in order to have the possibility of giving different selection probabilities to the boundary cells, but do not further elaborate on this aspect.

Finally, several mutation operators have been proposed that are intended to repair infeasible solutions which violate the problem constraints. The most basic repair mutation operator, is the random cell repair mutation (RCRM) (Cao, Batty, Huang, et al., 2011). The RCRM can be used to fix or reduce the magnitude a violation of the allowed number of cells per land-use type. The operator works similar to RCM, but now the land-use type of which a cell may be selected and converted to, are based upon the violation of the constraint. First of all, in case there are land-use types of which the number of cells are above the upper bound, only one of these may (randomly) be selected for mutation. Secondly, in case there are land-use types of which the number of cells are below the lower bound, the selected cells may only (randomly) be converted to one of these. The RCRM operator is visualised in Figure 4.16. The RCRM mutation operator can be applied multiple times, until the number of cells per land-use types are all within range again.

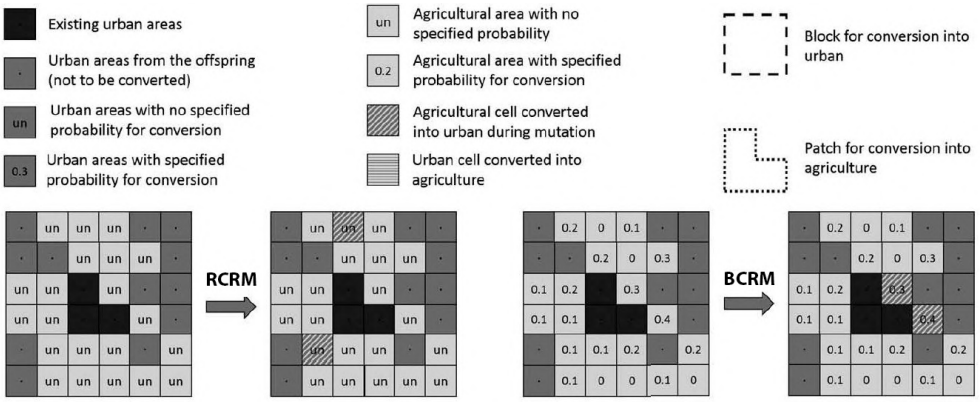


FIGURE 4.16: The RCRM and BCRM Operators (Schwaab, Deb, Goodman, et al., 2018)

A disadvantage of the RCRM operator, is the fact that it can easily fragment or create clusters, which is not beneficial for the compactness. Therefore, the biased cell repair mutation (BCRM) was introduced by (Schwaab, Deb, Goodman, et al., 2018). The procedure of BCRM is almost similar to the procedure of RCRM. However, the chance of a cell to be chosen is now higher if it has a lower compactness. As such, chances are higher that boundary cells are being converted, which is beneficial for the compactness. The procedure is visualised in Figure 4.16. The BCRM operator was described for a MOLA problem with two different land-use types. In case more land-use types are used, it can also be beneficial to count the number of neighbours having the same 'new' land-use type. As such, the chance of creating new 1-cell clusters can be reduced as well.

Another repair mutation operator, is the random patch repair mutation (RPRM) operator (Cao, Huang, Wang, et al., 2012). Similar to the RCRM operator, the RPRM operator can be used in case the allocation ranges constraint is violated. The operator works similar to RPM, but now the probability of the type and location of the path is biased. First of all, the probability of selecting a certain land-use type for the patch increases, when the number of cells of this type is currently below its lower bound. Secondarily, the probability of selecting a location located on a cell of certain type increases, when the number of cells of this type is above the upper bound. The research of (Cao, Huang, Wang, et al., 2012) does not specify how these probabilities are calculated in their implementation. The RPRM operator is expected to have a less negative effect on the compactness compared to the RCRM operator. However, in case the allowed range of cells per land-use type is rather small or even an exact number, converting complete patches might be an inefficient approach to eventually reach such a specific goal.

Next, a repair mutation operator based upon boundary cells was described in (Song and Chen, 2018). The operator, which we will call random boundary cell repair mutation (RBCRM), is a modified version of the previous discussed RBCM. The operator is shown in Figure 4.16 and works at follows. At first, all boundary cells are defined and a random boundary cell is selected. Next, the number of cells with a type equal to this boundary cell are counted. If this number is below the lower bound, the cell will not be converted to another type. If this number is above the upper bound, the cell will be converted to a random other land-use type (excluding the current type). In case the number is within its allowed range, the cell will be converted to a random type out of all types (including the current type). Similar to RBCM, defining all boundary cells can be computationally expensive.

The previous solutions all focus on repairing violations of the allowed number of cells per type. Another constraint that can be violated and repaired, is the minimum cluster size. In (Datta, Deb, Fonseca, et al., 2007), the MSIS2 operator was defined, which was inspired by the MSIS repair operator for class timetabling problem. The MSIS2 can be applied to a solution when the minimum cluster size constraint is violated, and works as follows. First, a naive search is executed to detect all boundary cells. For each of these boundary cells, the area of the corresponding cluster is calculated. In case the size of a cluster is below the minimum, neighboring cells (of this cluster) will be merged into this cluster. As can be seen, the MSIS2 operator can be a rather

computationally expensive operator. No further researches besides (Datta, Deb, Fonseca, et al., 2007) implements MSIS2 or any other operator for cluster size violation.

Problem Independent Vector Mutation

As far as we are aware, only one problem independent vector mutation for MOLA has been defined and applied in previous research. This was done by (Cao and Ye, 2013), which also implemented the previously discussed problem independent vector crossover operator SPX. The mutation operator the authors choose to apply to MOLA, was the single-point mutation (SPM) operator. The operator randomly selects one value (cell) in the vector, and changes its value (type). The SPM operator for vectors has the same result as the RCM mutation for grids. It was chosen by the authors for its simplicity.

Except for the SPM operator, other independent vector mutations could be applied to MOLA as well. Examples of such, are the swap and inversion mutation. The swap mutation simply swaps the values of two random values of the vector, whereas the inversion mutation inverses a random subset of the vector. Unfortunately, no research seems to implement any other operator besides SPM in the domain of MOLA.

Problem Dependent Vector Mutation

Again, no problem dependent vector mutation operator seems to have been defined or implemented in previous research. As was also the case with crossover operators, the amount of research towards vector-based mutation operators is relatively scarce. Again, this could be caused by the difficulty of obtaining spatial information corresponding to the values (regions) of the vector.

4.3.8 Single-Objective GAs for MOLA

Single-objective GAs have been applied to MOLA using a priori approaches in multiple researches. In this section, several of these application will be outlined. The focus will be on the eventual results that were obtained with these algorithms, in order to point out their advantages and disadvantages compared to other approaches.

Single-Objective Genetic Algorithm I

The first single-objective GA for MOLA that will be discussed, is from (Stewart, Janssen, and Herwijnen, 2004). The authors tend to minimize the costs, number of clusters per land-use and relative magnitude of the largest cluster, while maximizing compactness. The GA used goal programming as defined earlier in Equation (3.27), in combination with a penalty for exceeding bounds as was defined in Equation (3.28), (3.29) and (3.30). The GA implements a grid representation, and initializes uses Equation (4.10) and the two factors named afterwards. Selection is done according to fitness proportionate selection and crossover according to a problem dependent grid crossover operator that split clusters in half and switch their type. Finally, mutation was done using random block mutation (RBM). Eventually, the algorithm was applied to a MOLA problem with a grid of only 20x20 and 40x40. The

results focus mainly on the computational time of the GA. Hereby, the computational time appears to increase quadratically with problem size defined by the number of cells. Since it is eventually desired to solve MOLA with much larger grids than 40x40, more research towards the refinement of this algorithm is needed. According to the authors, one possibility would be to retain a relatively coarse resolution for initial exploration, and to increase the resolution for final tuning of the selected plans. Another option may be to simplify the spatial criteria and reduce the number of objectives. One year later in (Aerts, Van Herwijnen, Janssen, et al., 2005), the same authors of (Stewart, Janssen, and Herwijnen, 2004) extended their research with a more comprehensive comparison with an algorithm for MOLA based on simulated annealing. The general working of the single-objective simulated annealing algorithm is shown in Figure 4.1 and compared to the previous GA on a 20x20 MOLA problem with the same objectives (Figure 4.17). The results show that the GA generates slightly better results than simulated annealing (higher objective values). Especially considering the compactness values, the GA tends to win over the SA approach. The performance regarding the cluster size objective was slightly better for the SA algorithm.

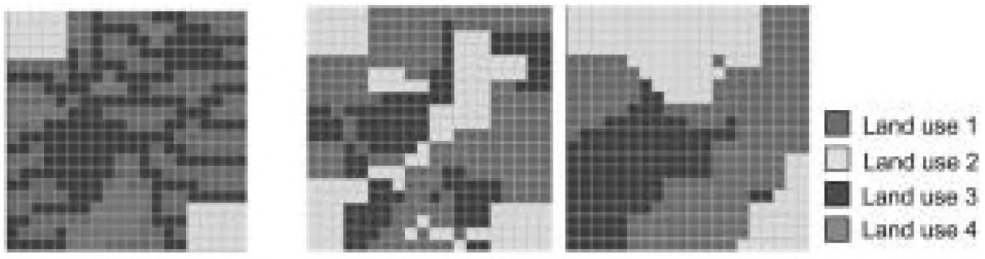


FIGURE 4.17: The original map (left), result of optimization with SA (centre) and result of optimization with GA (right) (Aerts, Van Herwijnen, Janssen, et al., 2005)

Single-Objective Genetic Algorithm II

The second single-objective GA that will be discussed, was proposed by (Cao and Ye, 2013). A vector representation was used, including a goal programming approach based on Wierzbicki as was defined in Equation (3.26). Initialization was done by creating random solutions and selection was done using fitness proportionate selection combined with elitism selection. Furthermore, crossover was done using single-point crossover as was shown in Figure 4.11 and mutation was realized using single-point mutation. Besides this single-objective algorithm (GGA), another version using parallelism was proposed and implemented as well (CGPGA). This coarse-grained type GA separates the chromosomes into several subsets to improve performance. Both algorithms were applied on a MOLA problem with a vector of size 586 and the objectives of maximizing the ecological suitability, accessibility and compatibility. The results of the single-objective GA were compared to the modified parallel version, as shown in Figure 4.18. With small population sizes (N=100), the performance of both were similar. When increasing the population (N=400), the parallel version became around 2.5 times as fast. As concerned the quality of the solutions provided by both algorithms, they were all near-optimal. Unfortunately, no further analysis, for example by comparing the algorithms to other available (grid based) approaches, was performed.

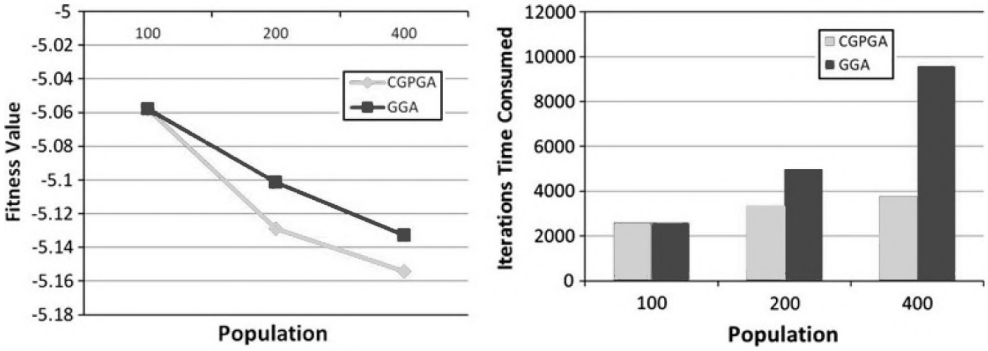


FIGURE 4.18: Comparison between GGA and CGPGA (Cao and Ye, 2013)

Single-Objective Genetic Algorithm III

The third single-objective GA for MOLA that will be discussed, was implemented in (Cao, Huang, Wang, et al., 2012). The GA was applied using the goal programming approach of Equation (3.26). A grid based representation was used, with randomized solutions as the initial population. Selection was done using fitness proportionate selection, and the boundary cell crossover operator (BCX-III) as shown in Figure 4.9 was applied for recombination. Mutation was done using random patch mutation (RPM and RPM-II) as visualized in Figure 4.14. Finally, the random patch repair mutation (RPRM) operator was implemented for reparation. The algorithm was applied to a MOLA problem with eight different objectives, among which all five objectives described in Chapter 3.3. Besides this, five different land-use types were present. At first, the run time of the algorithm was compared to a simple GA with basic problem independent operators. This GA took about 45.5h to finish the process, whereas the proposed GA only required 5.5h. Except for this comparison, the results mainly focus on the quality of solutions gained, discussing the scores obtained for each objective. According to the authors, the best solution found would be of such high quality that it could notably contribute to the actual planning process. However, no further comparisons between other approaches were given.

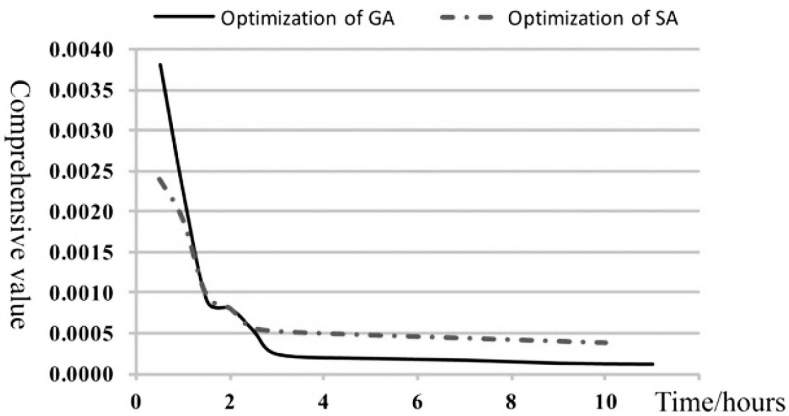


FIGURE 4.19: Comparison between GA and SA (Li and Parrott, 2016)

Single-Objective Genetic Algorithm IV

The final single-objective GA for MOLA that will be discussed, was proposed by (Li and Parrott, 2016). The GA implements a grid representation, and uses goal programming based on Wierzbicki for combining the objectives. Initialization is done using the knowledge-informed strategy that implements a selection value, as was described in Equation (4.11). Furthermore, selection is done using fitness proportionate selection and recombination is done using boundary cell crossover (BCX-IV) and two dimensional crossover operator (TDX). Finally, mutation was implemented using random cluster mutation (RCLM) and biased random cell mutation (BRCM). The algorithm was applied to a MOLA problem with three objectives, eight different land-use types and a grid size of 364 x 482 cells. The GA eventually took 9h to complete the process. This is a significant improvement compared to the 4.5h of (Cao, Huang, Wang, et al., 2012), which run with a grid size that was only one tenth of this one. However, the population size is only 20, which is considerably less than that required by the common GA. Eventually, the GA was also compared to an SA based approach. The comprehensive objective values over time for both SA and GA are shown in Figure 4.19. The results show that in the first 2h of computation, SA performed better than GA, with a comprehensive value that was smaller than that of GA. However, after the second hour, the rate of convergence of SA gradually reached 0; while our improved GA continued to evolve, resulting in a more ideal land-use scenario. Thus, while for our case study an SA would likely stop sooner, having converged towards a solution, the GA computes for a longer period of time and ultimately finds a more optimal land-use solution. This may be because the evolutionary operations, i.e., crossover and mutation, improve the ability of the GA to satisfy both the additive and spatial objectives. Still, as the authors mention, even with improved heuristics that increase the efficiency, there is still a gap between what the technology can provide and the computational demands of interactive land-use planning. Future research should for example focus on ways to enhance the efficiency of methods for MOLA. Heuristics such as GA and SA are currently insufficient.

4.3.9 Multi-Objective GAs for MOLA

The majority of algorithms that have been applied to MOLA are multi-objective GAs. In this section, several of these implementations will be outlined. For more details on the exact working of NSGA-II, view Chapter 4.3.5. In this section, the focus will be on the implementation of the algorithms together with the results that were obtained.

Multi-Objective Genetic Algorithm I

The first multi-objective GA for MOLA that will be discussed is from (Datta, Deb, Fonseca, et al., 2007), who introduced the 'NSGA-II-LUM' algorithm. The authors applied the algorithm on a three-dimensional MOLA problem as was discussed in Chapter 3.5.5. Initialization was done at random and two dimensional crossover operator (TDX) and boundary crossover operator (BCX) were used for crossover. Furthermore, random boundary cell mutation (RBCM) in combination with the boundary cell repair mutation (BCRM)

were used for mutation. Optionally, an (computationally expensive) guidance/repair procedure could be used to reduce the violation of (initial) solutions. The objectives are merged into a single weighted objective in the following manner:

Minimize:

$$F(x) = \sum_{i=1}^{\mathcal{M}} \bar{w}_i^x f_i(x) \tag{4.14}$$

where $\mathcal{M}$ equals the number of objective functions, and weight $\bar{w}_i^x$ (of objective i) is defined as:

$$\bar{w}_i^x = \frac{(f_i^{max} - f_i(x))/(f_i^{max} - f_i^{min})}{\sum_{j=1}^{\mathcal{M}}((f_j^{max} - f_j(x))/(f_j^{max} - f_j^{min}))} \tag{4.15}$$

hereby f_i^{min} and f_i^{max} are equal to the minimum and maximum objective values of objective i that have been found. After all weights have been calculated, a local search is performed. The local search operators used in NSGA-II-LUM of (Datta, Deb, Fonseca, et al., 2007), replaced every boundary cell by one of its adjacent cells (in case the constraints allows so). Then $F(x)$ is re-evaluated, and the change (for each boundary cell) is accepted if an improvement in $F(x)$ is found.

The results of (Datta, Deb, Fonseca, et al., 2007) showed interesting insights regarding the feasibility of solutions. The algorithm was applied to a MOLA problem with a 100x100 grid and 5 different land-use types. One of the constraints concerned a minimum and maximum cluster size. Without any special guidance, the NSGA-II was not able to produce any results that satisfied this constraint. First of all, all solutions in the initial population (generated at random) violated the problem constraints. Secondly, even after 5000 generation (where the the BCRM repair operator was also active), there was no feasible solution among the final results either. As such, a special guidance procedure had to be applied, which increased the computational complexity of the process significantly. For more details on this procedure, view (Datta, Deb, Fonseca, et al., 2007). Next up, a comparison was made between the problem independent crossover operator TDX and the problem dependent crossover operator BCX. The results are shown in Figure 4.20, where all three objectives needed to be minimized. Remarkably, using the problem independent TDX operator resulted in higher quality solutions. A test was performed to discover whether these bad results were caused by the mutation ratio of BCX. However, with every mutation ratio (0-100%) for BCX, the TDX still delivered better results. Finally, no further comparisons were made between NGSA-II-LUM and other algorithms for MOLA.

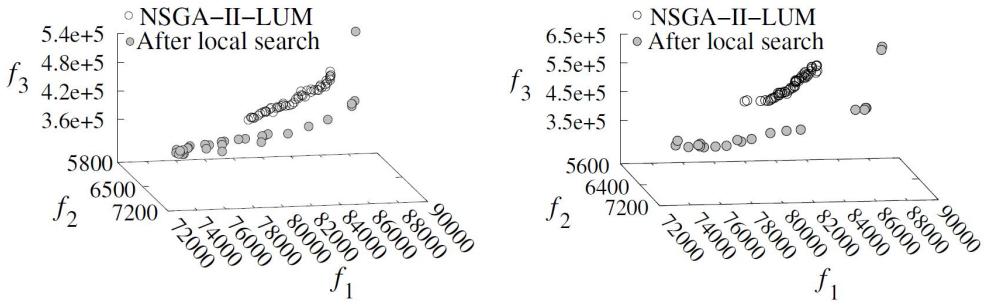


FIGURE 4.20: NSGA-II-LUM using TDX, RBCM and BCRM (left) and BCX, RBCM and BCRM (right) (Datta, Deb, Fonseca, et al., 2007)

Multi-Objective Genetic Algorithm II

The second application of NSGA-II to MOLA that will be discussed, is the 'knowledge-improved' NSGA-II (KI-NSGA-II) algorithm introduced by (Schwaab, Deb, Goodman, et al., 2018). According to these authors, the KI-NSGA-II differs from NSGA-II on three different aspects. First of all, the 'archive' size was being limited. The archive contains the best (non-dominated) solutions that have been found. The archive is updated after every generation, and its size can be become quite large. Limiting the size to the population size by truncation non-dominated solutions with the lowest crowding distance when this size is exceeded, could so prevent this from happening. Eventually, this could increase the performance of NSGA-II. Secondly, a knowledge-informed initialization strategy is being used; instead of randomly generating solutions, solutions are now generated by mutating 30% of the current situation. Finally, three knowledge-informed operators are being used; the boundary cell crossover operator BCX-II, the random patch mutation operator RPM-III and the random boundary cell repair mutation operator RBCRM.

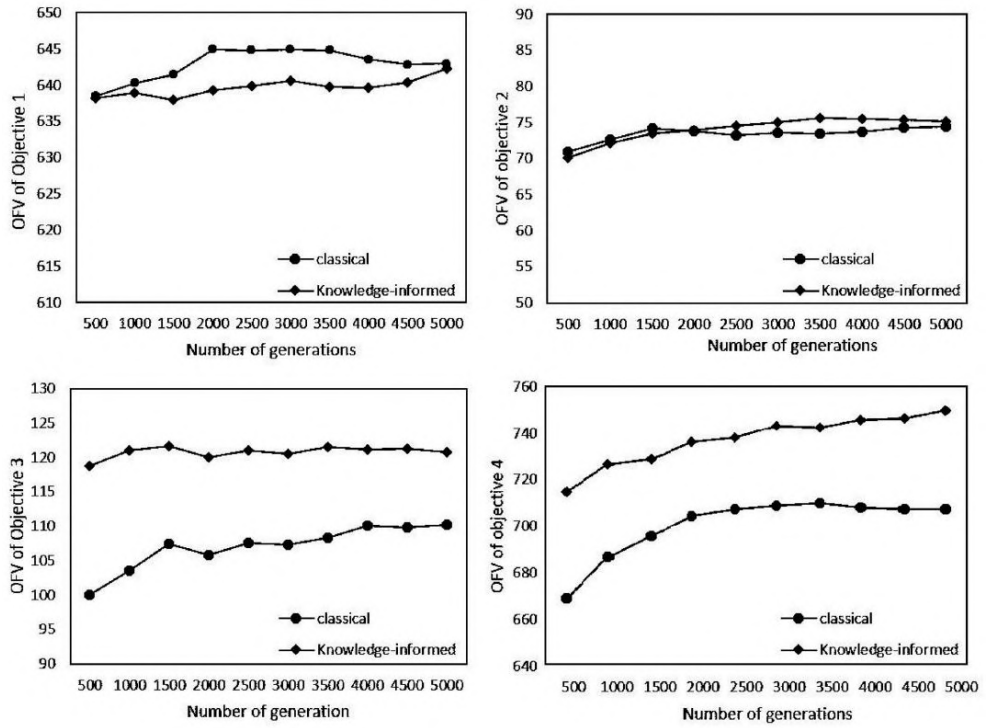


FIGURE 4.21: OFV Values of NSGA-II and KI-NSGA-II (Song and Chen, 2018)

Both NSGA-II and KI-NSGA-II were applied to a MOLA problem on a 30x30 grid with three different land-use types. Four objectives were optimized; maximization of the suitability for agricultural, construction and conservation (obj. 1 to 3) and maximization of the overall compactness (obj. 4). The performance of the algorithms were measured, by plotting the the average OFVs of the solutions for the four objectives, as shown in Figure 4.21. The OFV value is based upon the ARI and ACD values, which are measure for respectively the closeness to a the Pareto front and the diversity of the solutions. View (Schwaab, Deb, Goodman, et al., 2018) for the exact calculation of both both values. As a lower OFV value is better, the KI-NSGA-II does not perform any better regarding obj. 1 and 2, but it is obviously more efficient in handling obj. 3 and 4. The improvement in obj. 3 can be explained by the fact that land that is to be conserved, is strictly protected from being converted during the initialization and the operators used. The improvement in obj. 4, the compactness, is most likely caused by the fact that both the crossover and mutation operators are designed to encourage compact land allocation, in contrast to the operators of the classical NSGA-II algorithm.

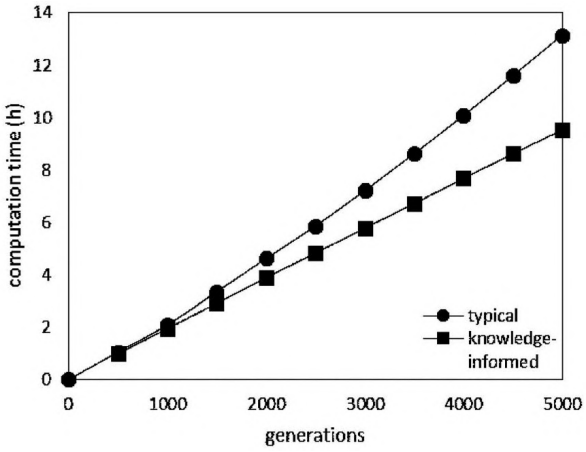


FIGURE 4.22: Computational Time of NSGA-II and KI-NSGA-II (Song and Chen, 2018)

As can be seen in Figure 4.22, when the number of generations increases, the computation time of KI-NSGA-II increases at a lower rate than the computation time of NSGA-II. Eventually, NSGA-II took 13 hours and 11 minutes to complete 5000 generations whereas the KI-NSGA-II algorithm took only 9 hours. Together with the previously compared OFV scores, we can so conclude that KI-NSGA-II performs better than NSGA-II.

Multi-Objective Genetic Algorithm III

The final multi-objective GA for MOLA that will be discussed, is the so-called NSGA-II-MOLU algorithm from (Cao, Batty, Huang, et al., 2011). The algorithm uses a grid representation, with an initial population existing out of random solutions (90%) and solutions with the current land-use allocation (10%). Remarkably, the algorithm does not implement crossover; instead, it implements the random patch swap mutation (RPSM) operator as a 'single-parent' crossover operator. Furthermore, the random patch mutation (RPM) and random cell repair mutation (RCRM) are used. The algorithm was applied to a 400x400 grid with 5 different land-use types. Three objectives were optimized; minimization of the conversion costs, maximization of the accessibility, and maximization of the compatibility. The authors claim the algorithm to be a step forwards in terms of speed. The use of these three (mutation) operator, made the algorithm require less than 10 minutes of computation for 1000 generations with a population of size 100 on a standard (ca. 2010) general PC. The authors state this proves that single-parent crossover (RPSM) operator is superior to traditional two-parent crossover. However, no actual comparison with an algorithm implementing two-parent crossover was done to prove this claim.

Chapter 5

Pareto Local Search

In this chapter, a small literature study towards the Pareto Local Search algorithm will be conducted. This will be succeeded by several variations on this algorithm, such as multi-start Pareto local search, iterated Pareto local search and genetic Pareto local search. Finally, the possibilities of anytime behavior, including several neighborhood search strategies, will be handled.

5.1 Definition

Pareto Local Search (PLS) is a simple iterative improvement algorithm for multi-objective combinatorial optimization problems. It was first proposed and experimentally tested in (Paquete, Chiarandini, and Stützle, 2004) for the multi-objective traveling salesman problem. Independently, a similar algorithm was proposed by (Angel, Bampis, and Gourvés, 2004). Pareto Local Search extends the single-objective hill climbing algorithm to make it suitable for multi-objective problems as well (Cabrera-Guerrero, Mason, Raith, et al., 2018). In PLS, an archive is used to store the best (non-dominated) solutions that have been found so far. At the beginning, the archive only holds the initial solution. Next, local search operators are applied to solution(s) in the archive, to obtain new solutions. Solutions that are non-dominated by the solutions in the archive are added. At the same time, solutions from the archive that become dominated are removed. As so, the PLS algorithm continuously improves upon a set of solutions, instead of a single one.

Algorithm 4 PLS ($\mathcal{A}_0$) (Paquete, Chiarandini, and Stützle, 2004)

input: The initial set of non-dominated solutions $\mathcal{A}_0$
explored(s) := FALSE $\forall s \in \mathcal{A}_0$
$\mathcal{A} := \mathcal{A}_0$
while $\mathcal{A}_0 \neq \emptyset$ **do**
    s := select random solution from $\mathcal{A}_0$
    for all $s' \in \mathcal{N}(s)$ **do**
        if $\mathcal{A} \not\prec s'$ **then**
            explored(s') := FALSE
            $\mathcal{A}$:= Update($\mathcal{A}$,s')
        end if
    end for
    explored(s) := TRUE
    $\mathcal{A}_0 := \{s \in \mathcal{A} \mid \text{explored}(s) = \text{FALSE}\}$
end while
output: $\mathcal{A}$

Algorithm 4 illustrates the PLS framework (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015). The PLS algorithm is called with a set $\mathcal{A}_0$ that contains (non-dominated) starting solutions. All of these are given the status 'unexplored'. Now, PLS continuously keeps record of an archive $\mathcal{A}$, to which all solutions of $\mathcal{A}_0$ are initially added. Next, the algorithm iteratively picks a solution s from the unexplored solutions of $\mathcal{A}$ and explores its neighborhood. All solutions found that are non-dominated by all solution of $\mathcal{A}$, are added to $\mathcal{A}$. At the same time, solutions that now get dominated are removed from $\mathcal{A}$. The updating procedure is responsible for this process, and is shown in Algorithm 5. After all solutions of the neighborhood of s have been examined, s is given the status 'explored'. PLS terminates after are solutions in $\mathcal{A}$ have received this status. The final archive $\mathcal{A}$ is the outcome (Pareto front approximation) of the algorithm.

Algorithm 5 Update($\mathcal{A}$, $\mathcal{A}'$) (Paquete, Chiarandini, and Stützle, 2004)

1: **input:** A set of non-dominated solutions $\mathcal{A}$ and a set $\mathcal{A}'$
2: $\mathcal{A}'' := \emptyset$
3: **for all** $s \in (\mathcal{A} \cup \mathcal{A}')$ **do**
4: **if** $s' \in (\mathcal{A} \cup \mathcal{A}') \not\prec s$ **then**
5: $\mathcal{A}'' := \mathcal{A}'' \cup s$
6: **end if**
7: **end for**
8: **output:** $\mathcal{A}''$

The PLS algorithm can be decomposed in four main algorithmic components that need to be defined before running the algorithm (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015).

- **Input initialization.** Although this is officially not part of the algorithm, this component defines the input of the algorithm. Note, that it is possible to start of with a single solution (for example the current land-use allocation) as input $\mathcal{A}_0$.
- **Selection procedure.** This procedure is responsible for selecting a solution from the archive of which the neighborhood will be explored next. In the standard PLS algorithm, the (yet unexplored) solution is chosen from the archive at random.
- **Acceptance criterion.** This procedure is responsible for determining whether or not a new solution may be added to the archive or not. In the standard PLS algorithm, a solution may be added to archive in case it is not dominated by any other solution currently in the archive.
- **Neighborhood exploration.** This procedure is responsible for exploring the neighborhood of a solution. In the standard PLS algorithm, this is done by examining the complete neighborhood. Other strategies will be discussed later in Chapter 5.4.

The standard version of PLS has proven to be an efficient technique for many multi-objective optimisation problems. Over time, different variants of the PLS have arisen over time to improve on the performance. The Multi-Restart PLS, Iterated PLS and Genetic PLS will now be discussed in Chapter 5.2.1,

5.2.2 and 5.2.3. Finally, PLS is said to have poor 'anytime behavior', stating that it can take up to a long time before $\mathcal{A}$ contains a (relatively) qualitative approximation of the Pareto front. In Chapter 5.4, we will discuss several possible improvements for the anytime behavior of PLS. This will, among other extensions, include a variety of possible neighborhood exploration strategies.

5.2 Variants

5.2.1 Multi-Restart Pareto Local Search

The first variant of PLS that will be discussed, is the multi-restart Pareto local search (MPLS). This algorithm tends to prevent getting stuck in a local optimum or minumum, by iteratively restarting the PLS algorithm with a new random initial population. In this section, a variant on the 'standard' MPLS algorithm will be discussed, that uses a set of non-dominated solutions as the new initial population, instead of only one (random) solution. The pseudocode for the algorithm is shown in Algorithm 6 (Drugan and Thierens, 2012).

Algorithm 6 Multi-Restart PLS ($\mathcal{T}$) (Drugan and Thierens, 2012)

input: A stopping criterion $\mathcal{T}$
$\mathcal{A} := \emptyset$
while Stopping criterion $\mathcal{T}$ is not met **do**
    s := generate random solution (uniformly)
    $\mathcal{A}'$:= Deactivate(s, $\mathcal{A}$)
    $\mathcal{A}$:= Update($\mathcal{A}$, Pareto Local Search($\mathcal{A}'$))
end while
output: $\mathcal{A}$

The input parameter for MPLS is its own stopping criterion. This can, for instance, be the maximum number of restarts. At the start of each restart, a random solution s is generated. Given the deactivation procedure of Algorithm 7, this solution is used to create a Pareto set out of $\mathcal{A}$ that will serve as the input for the next PLS run. This Pareto set exists out of s and all solutions of the archive that do not dominate nor are being dominated by s. Finally, the outcome of the PLS algorithm is used to update archive $\mathcal{A}$.

Algorithm 7 Deactivate(s, $\mathcal{A}_0$) (Drugan and Thierens, 2012)

input: An set of non-dominated solutions $\mathcal{A}_0$
$\mathcal{A} := \{s\}$
for all $s' \in \mathcal{A}_0$ **do**
    if $s \not\prec s'$ and $s' \not\prec s$ **then**
        $\mathcal{A}$:= Update($\mathcal{A}$, s')
    end if
end for
output: $\mathcal{A}$

The difference between the MPLS of Algorithm 6 and the basic MPLS of (Paquete, Chiarandini, and Stützle, 2004), is the use of the deactivation algorithm. However, as most solutions from $\mathcal{A}$ will most likely dominate s, the procedure will probably still lead to a set with only s (and so by similar to

standard MPLS). As so, the deactivation procedure will most likely only be profitable in case relatively fit new solutions can be created.

5.2.2 Iterated Pareto Local Search

The second variant of PLS that will be discussed, is Iterated Pareto Local Search (IPLS). In Algorithm 8, the pseudo-code of an IPLS based upon (Drugan and Thierens, 2010) is shown. Given a stopping criterion $\mathcal{T}$, IPLS works as follows. At first, the MPLS is run to construct an initial archive $\mathcal{A}$. Then, a solution is chosen from the archive at random and mutated. This mutated solution then forms the input for the PLS algorithm. This continues until criterion $\mathcal{T}$ is satisfied.

Algorithm 8 Iterated PLS ($\mathcal{T}$) (Drugan and Thierens, 2010)

1: **input:** An set of non-dominated solutions $\mathcal{A}_0$ and a stopping criterion $\mathcal{T}$
2: $\mathcal{A}$:= Multi-Restart Pareto Local Search ($\mathcal{A}_0$)
3: **while** Stopping criterion $\mathcal{T}$ is not met **do**
4: s := select random solution from $\mathcal{A}$
5: s' = Mutate(s)
6: $\mathcal{A}$:= Update($\mathcal{A}$, Pareto Local Search (s'))
7: **end while**
8: **output:** $\mathcal{A}$

Optionally, the IPLS algorithm can be combined with the meta-heuristic method Variable Neighborhood Search (VNS) (Geiger, 2011). The pseudo-code for this algorithm is shown in Algorithm 9. The search starts with a single, randomly generated solution. Then, the neighborhood of the current solution is generated, updating the approximation set $\mathcal{A}$ of the Pareto front. Line 10 of the pseudo-code then distinguishes two cases: Either the current solution is replaced by some neighboring, dominating one, or the search continues with the next neighborhood operator. In the first case, more than one neighboring solution might exist that dominates the current one. This implies that the selection of a replacing solution needs to be made, which, in the following, we implement by a random choice. The entire process continues until all elements of $\mathcal{A}$ have been searched using the neighborhoods $N_1, ..., N_k$. Escaping local optima is then tried by the use of some mutation operator, stated in lines 23–25. For now, we will not go further into detail on the combination of IPLS and VNS. View (Geiger, 2011) or (Geiger, 2008) for more information regarding its working and performance.

Algorithm 9 Iterated PLS with VNS ($\mathcal{T}$) (Geiger, 2011)

input: A stopping criterion $\mathcal{T}$
Initialize control parameters: Define the neighborhoods $N_1, ..., N_k$
$i := 1$
s := generate random solution (uniformly)
$\mathcal{A} := \{s\}$
while Stopping criterion $\mathcal{T}$ is not met **do**
    while s locally optimal with respect to $N_1, ..., N_k$, therefore $i > k$ **do**
        $\mathcal{A}$:= Update($\mathcal{A}, \mathcal{N}_i(s)$)
        if $\exists s' \in \mathcal{N}_i(s) \mid s' \preceq s$ **then**
            $s := s'$
            $i := 1$
            Rearrange the neighborhoods $N_1, ..., N_k$ in a random order
        else
            $i := i + 1$
        end if
    end while
    explored(neighborhoods of s) := TRUE
    $i := 1$
    if $\exists s' \in \mathcal{A} \mid$ neighborhoods not investigated yet **then**
        $s := s'$
    else
        s' := select random $s' \in \mathcal{A}$
        s'' := Mutate(s')
        $s := s''$
    end if
end while

5.2.3 Genetic Pareto Local Search

The final variant of PLS that will be discussed, is genetic Pareto local search (GPLS). The GPLS algorithm, as derived from (Drugan and Thierens, 2012), is shown in Algorithm 10. First of all, the initial archive $\mathcal{A}$ is created using MPLS. Next, a random solution is chosen from $\mathcal{A}$ and either mutated or recombined with another random solution from the archive. The α value determines what part of the solution gets mutated and what part gets recombined. In general, mutating a solution results in less different ('closer') solution than when recombining. Next, the deactivation procedure is used again to create a starting set for PLS given the mutated or recombined solution. This goes on until criterion $\mathcal{T}$ is satisfied.

Algorithm 10 Genetic PLS (α, $\mathcal{T}_1$, $\mathcal{T}_2$) (Drugan and Thierens, 2012)

input: An initial set of non-dominated solutions $\mathcal{A}_0$, a stopping criterion $\mathcal{T}_1$ and a stopping criterion $\mathcal{T}_2$
$\mathcal{A}$:= Multi-Restart Pareto Local Search ($\mathcal{T}_1$)
while Stopping criterion $\mathcal{T}_2$ is not met **do**
    s := select random solution from $\mathcal{A}$
    if $\alpha > \mathrm{U}(0,1)$ or $|\mathrm{A}| < 2$ **then**
        s' := Mutate(s)
    else
        s'' := select random solution $s'' \neq s$ from $\mathcal{A}$
        s' := Recombine(s, s'')
    end if
    $\mathcal{A}'$:= Deactivate(s', $\mathcal{A}$))
    $\mathcal{A}$:= Update($\mathcal{A}$, Pareto Local Search ($\mathcal{A}'$))
end while
output: $\mathcal{A}$

5.3 Components

5.3.1 Selection Procedures

In the standard PLS algorithm, the next solution to be explored is selected at random. A different method, is selecting the solution that is expected to deliver the best improvements. This can be done using Optimistic Hypervolume Improvement (OHI), shown in Figure 5.1 (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015). The OHI value indicated the 'gap' between a solution and its neighboring solutions. A bigger gap can be seen as a less explored neighborhood, which is more interesting for the PLS algorithm. The OHI of solution s is calculated as follows;

$$\mathrm{OHI}(s) = \begin{cases} 2\mathrm{ohvc}(s_{sup}, s) & \text{if } \nexists s_{inf} \\ 2\mathrm{ohvc}(s, s_{inf}) & \text{if } \nexists s_{sup} \\ \mathrm{ohvc}(s_{sup}, s) + \mathrm{ohvc}(s, s_{inf}) & \text{otherwise} \end{cases} \tag{5.1}$$

where;

$$s_{sup} = \mathrm{argmin}_{s_i \in \mathcal{A}}\{f_2(s_i) | f_2(s_i) > f_2(s)\} \tag{5.2}$$

$$s_{inf} = \mathrm{argmax}_{s_i \in \mathcal{A}}\{f_2(s_i) | f_2(s_i) < f_2(s)\} \tag{5.3}$$

and in case of the bi-objective space;

$$\mathrm{ohvc}(s, s') = (f_1(s) - f_1(s')(f_2(s') - f_2(s) \tag{5.4}$$

The s_{sup} and s_{inf} are the two closest neighbours of s. In case either s_{sup} or s_{inf} does not exist (when s is an extreme), the OHI value equals double the gap with its only neighboring solution. The OHI value can be calculated in linear time when optimizing two objectives. When optimizing more objectives, a

data structure will be required that allows for an efficient search towards a solution's nearest solutions (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015). As an alternative to random selection, PLS can now select the solution with the highest OHI value.

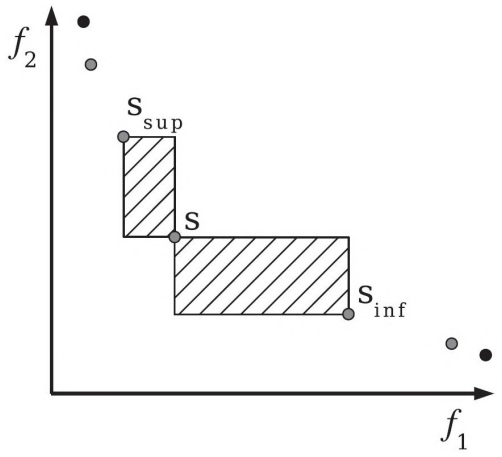


FIGURE 5.1: The OHI is the total area (marked) between a solution its closest neighbors (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015)

5.3.2 Acceptance Criteria

The standard PLS algorithm accepts all solutions that are not being dominated by any solution from the archive. Different criteria can be applied, for example more strict ones, to prevent the archive from growing too fast. An example of such a criterion, is to only accept solutions that dominate the solution being explored. This could result in a faster convergence, but also decrease the overall approximation quality. Finally, we can also switch from one rule to another during one neighborhood exploration. Meaning, that if one or more neighborhood solutions dominate the solution being explored, only these are accepted. If no neighborhood solutions dominate the solution being explored, other solutions (if non dominated by the archive of course) are accepted as well.

5.3.3 Neighborhood Exploration

Finally, standard PLS explores all neighboring solutions. The exploration strategy influences both the quality of the outputted Pareto set and its size. In (Drugan and Thierens, 2012) three neighbourhood exploration strategies are described that can be applied to PLS. Such an improvement strategy can be passed on as a parameter to a local search algorithm (noted as $\mathcal{I}$ in (Drugan and Thierens, 2012)). Finally, a type of order relationship and a fitness function are required in order to realize a search strategy. We assume both are constantly present, and have left out the fitness function notation in the pseudo-code for the sake of simplicity.

Algorithm 11 Best Pareto Improvement (s, $\mathcal{A}$, $\mathcal{N}$) (Drugan and Thierens, 2012)

1: $\mathcal{A}' := \{s\} \cup \mathcal{A}$
2: **for all** $s' \in \mathcal{N}(s)$ **do**
3: **if** $\forall s'' \in \mathcal{A}', s' \prec s'' \vee s' || s''$ **then**
4: explored(s') := FALSE
5: $\mathcal{A}'$:= Update($\mathcal{A}'$,{s'})
6: **end if**
7: **end for**
8: $\mathcal{A}' := \mathcal{A}' \setminus (\{s\} \cup \mathcal{A})$
9: **output:** $\mathcal{A}'$

The first strategy that will be discussed is Best Pareto Improvement (BPI), of which the pseudo-code is given in Algorithm 11 (Drugan and Thierens, 2012). This search strategy examines each solution of the neighborhood that is being explored. The resulting archive $\mathcal{A}$ contains all solutions of the neighborhood that are not dominated by either s or the current archive $\mathcal{A}$ (or each other). Possibly, set $\mathcal{A}$ is empty, meaning no non-dominated solutions were found.

Algorithm 12 Neutral Pareto Improvement (s, $\mathcal{A}$, $\mathcal{N}$) (Drugan and Thierens, 2012)

1: $\mathcal{A}' := \{s\} \cup \mathcal{A}$
2: **for all** $s' \in \mathcal{N}(s)$ **do**
3: **if** $\forall s'' \in \mathcal{A}', s' \prec s'' \vee s' || s''$ **then**
4: explored(s') := FALSE
5: $\mathcal{A}'$:= Update($\mathcal{A}'$,{s'})
6: $\mathcal{A}' := \mathcal{A}' \setminus (\{s\} \cup \mathcal{A})$
7: **output:** $\mathcal{A}'$
8: **end if**
9: **end for**
10: **output:** $\emptyset$

The second search strategy that will be discussed, is called Neutral Pareto Improvement (NPI) (Drugan and Thierens, 2012). The pseudo-code is given in Algorithm 12. The procedure simply terminates as soon as one solution is found that is not dominated by the solution being explored and all solutions in the archive. An empty set is returned in case no solution is found in the entire neighborhood that does so.

Algorithm 13 First Pareto Improvement (s, $\mathcal{A}$, $\mathcal{N}$) (Drugan and Thierens, 2012)

```
1: A' := {s} ∪ A
2: for all s' ∈ N(s) do
3:     if ∀s'' ∈ A', s' ≺ s'' ∨ s' || s'' then
4:         explored(s') := FALSE
5:         A' := Update(A',{s'})
6:         if s' ≺ s then
7:             A' := A' \ ({s} ∪ A)
8:             output: A'
9:         end if
10:    end if
11: end for
12: A' := A' \ ({s} ∪ A)
13: output: A'
```

Finally, the pseudo-code of the First Pareto Improvement (FPI) is given in Algorithm 13. In FPI, the procedure terminates as soon as a solution is found that dominates the solution of which the neighborhood is being explored. For now, we will not go into further detail on the different exploration strategies. View (Drugan and Thierens, 2012) for a more elaborate view on all definitions and proofs related to the discussed strategies, including a fourth strategy based on hypervolume.

5.4 Anytime Behavior

The standard PLS of Algorithm 4 terminates when each solution of the archive is marked as 'explored'. However, this can result in very long running times (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015). Possibly, it can take up to an exponential time to eventually terminate, as the size of the archive is not restricted. When ran on real life problems of common sizes, this could result in running times of multiple days. As so, it can be desired to terminate the PLS algorithm beforehand, and still obtain a high quality approximation. A so-called 'anytime algorithm' aims to realize such a qualitative approximation as fast as possible, and continues on improves upon this quality for as long as the algorithm runs. As so, anytime algorithm can be terminated 'at any time' and are still expected to return a good approximation.

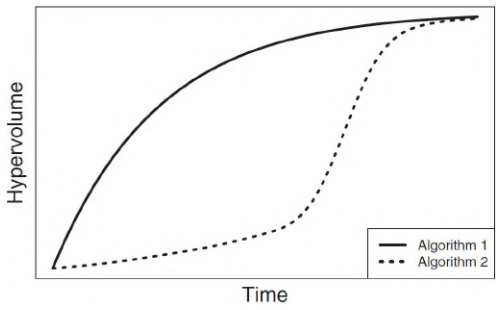


FIGURE 5.2: Example of an algorithm with good anytime behavior (1) and bad anytime behavior (2) (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015)

The standard PLS algorithm delivers a poor approximation quality if terminated before completion. In order to measure the approximation quality of a Pareto front, the hypervolume indicator can be used. The hypervolume of a set of solutions is equal to the area of the objective space that those solutions dominate (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015). In order to compute this area, a reference point is used that is dominated by each of those solutions. A higher hypervolume corresponds to a better approximation quality, which is shown in Figure 5.2. In (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015), the influence of different components on the hypervolume of the archive over time was compared to the original PLS. The results show for instance, that the use of an OHI based selection clearly improves the anytime behavior compared to the standard selection procedure. Secondly, the FPI strategy tends deliver a better any time behavior than the standard BPI strategy. However, combining different components, could again degrade the performance, showing that these components interact with each other. For more details on all test results obtained, view (Dubois-Lacoste, López-Ibáñez, and Stützle, 2015).

Phase II

Implementation and Experimentation

Chapter 6

Pareto Local Search for MOLA

6.1 Introduction

In this chapter, the Pareto Local Search algorithm for MOLA including several knowledge-informed components and operators will be defined. The algorithms are build upon previous research, further extending the PLS procedures to specifically fit the MOLA problem. By exploiting common properties and domain knowledge of MOLA, these extensions aim to improve the efficiency of the optimization process in this specific domain. All algorithms, components and operators will be outlined in detail and the (important) extensions being proposed will be justified.

In Chapter 6.2, the setup of the algorithm will be discussed. This will include the objectives and constraints on which the proposed algorithm will mainly focus, and the problem representation and data structures that will be used to efficiently store and retrieve the information necessary. As the algorithm is loosely coupled, it can easily be extended to handle a different setup as well. In Chapter 6.3, the algorithms of Pareto Local Search for MOLA (PLS-MOLA) and Iterated Pareto Local Search for MOLA (IPLS-MOLA) will be defined. These are modified versions of the general (iterated) PLS algorithm, specifically focusing on the MOLA domain. Next up, several components and/or procedures that can be implemented in this algorithm will be handled. First, the selection procedure including two selection operators for selecting the next solution from the archive, will be discussed in Chapter 6.4 and 6.5. This will be followed by two different neighborhood exploration strategies for exploring the selected solution in Chapter 6.6. In order to do so, a general neighborhood search procedure and three possible search operators will be proposed in Chapter 6.7 and 6.8. Next, the general reparation procedure and two reparation operators to repair solutions that violate constraints will be formulated in Chapter 6.9 and 6.10. This will be followed by three criteria that can be used for accepting solutions in the archive in Chapter 6.11. Finally, incremental metadata updating and constraint validation will be discussed in Chapter 6.12 and 6.13 and a perturbation procedure for IPLS will be proposed in Chapter 6.14.

6.2 Algorithm Setup

In order to design efficient knowledge-informed algorithm components, we will first have to define what (types of) objectives and constraints will be taken in account. For now, we will focus on the most common ones in previous literature, that support both incremental checking and updating (this will be explained later). Extensions to these can easily be set up in future search.

Finally, we will decide upon an appropriate problem representation and data structure that allow for an efficient solution storage and optimization process.

6.2.1 Problem Specification

Many objectives can be optimized within a MOLA problem. In Chapter 3.3, five of these were discussed. The Tongzhou case study of Chapter 3.6.3 even incorporated eight different objectives. Besides objectives, we can also apply multiple constraints, of which several were discussed in 3.2. It is out of the scope of this project to design efficient optimization procedures and components for all (types of) objectives and constraints. This section will therefore narrow down the MOLA problem to a more concrete case, on which the PLS for MOLA algorithm will focus.

The objectives for MOLA can take on many different forms and types, that may require different approaches to realize an efficient optimization process. When applying PLS to MOLA, there is one type of objective that is highly preferable; objectives of which the value is incrementally adaptable. This means, that when a small change has been applied to a solution, it should not be necessary to use the entire solution to recalculate the new objective value. Knowing the changed cell, and possibly its neighbors, should allow you to calculate the (relative) change in objective value. As the PLS algorithm constantly applies a lot of small changes, it would otherwise require the algorithm to continually recalculate the complete objective value, which can be costly. Therefore, this research will require the objective functions to be incrementally updatable. As so, only objectives that satisfy Definition 6.2.1 will be accepted.

Definition 6.2.1 (Incremental Objective)**.** An objective o is considered as incremental, if there exists a function $f_o(s)$ that can compute the objective value of o of a solution s created from a solution s' in $O(n)$ time, with n being the number of land-use changes between s' and s.

Definition 6.2.2 (Incremental Constraint)**.** A constraint c is considered as incremental, if the validation of a solution s created from a valid solution s' subject to constraint c can be done in $O(n)$ time, with n being the number of land-use changes between s and s'.

The incremental objective o should be able to calculate the objective value of a newly created solution based upon the applied changes and previous objective value. In case n changes have been applied, calculating this new value should cost at most $O(n)$ time. In most cases, such an objective function is formulated as $f_o(s) = \sum_{c \in S} f_o^c(c)$, where $f_o^c(c)$ is the objective value related to one single cell c of solution s. Besides objective value calculation, constraint checking is also an operation that needs to be performed each time a new solution has been formed. As such, the validity of a solution will also be required to be verifiable in an incremental manner. The definition of an incremental constraint is given in Definition 6.2.2. Similar to objective calculation, a solution that was created due to n changes on a another valid solution, should be validatable in at most $O(n)$ time.

Although we have now required the objective functions and constraints to be incrementally updatable, there are still a large number of objectives that can be optimized. As it is out of the scope of this project to design efficient

optimization procedures and components for all of these objectives and constraints, only the most common ones will be taken in account. In this case, the algorithm will optimize the general MOLA problem as defined in Chapter 3.2, using the following two objectives and constraint. The codes upfront will later on be used for referencing.

Objective O1: Minimizing Development Costs. The most basic and often used objective is the minimization of development costs. In case of the single-objective land-use allocation problem, this is often the (only) objective. The exact definition of this objective is given in Equation (3.9).

Objective O2: Maximizing Compactness. The second most used objective is the maximization of the compactness. The definition of this objective is given in Equation (3.13). The algorithm will focus on the four-neighbour variant, which is the most simple form.

Constraint C1: Allocation Ranges. Finally, the most often used constraint will be taken in account, which is an allowed range on the number of cells per land-use type. The exact definition of this constraint is given in Equation (3.6).

As noted earlier, the PLS algorithm for MOLA can be used to optimize other objectives and constraints as well as the algorithm is loosely coupled. Therefore, other operators or procedures can be implemented easily. Finally, a solution can contain both dynamic or static land-use types, whereas static land-use types are not allowed to change. It is required that each MOLA problem should have at least two dynamic land-use types, in order to be optimizable.

6.2.2 Problem Representation

In Chapter 3.5, several possible problem representations including their advantages and disadvantages have been discussed. In this research, the choice has been made to work with the grid representation of Chapter 3.5.1. The grid representation is the most straightforward and convenient representation to work with. Although the vector representation can reduce redundancy, these structures require additional spatial information in order to for example support spatial operators or fitness functions, which can significantly increase the complexity. Besides, almost all previous research makes use of the grid based representation, including the operators defined in Chapter 4.3.6 and 4.3.7. When researching PLS for MOLA, we aim to build upon previous research findings as much as possible. Usage of a relatively little researched representation such as a vector would require a lot more investigation, which is out of the scope of this research. Therefore, this research will implement the grid representation, which is shown in Figure 6.2.3 In this representation, a land-use type is stored in each cell of the grid. More details on this representation can be found in Chapter 3.5.1.

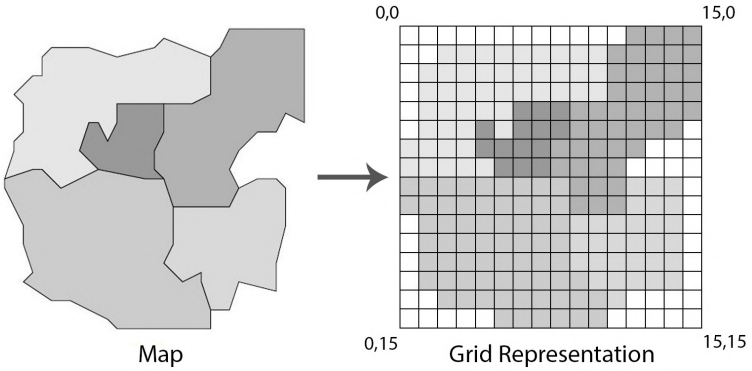


FIGURE 6.1: 16x16 Grid Representation (Matthews, Craw, Elder, et al., 2000)

6.2.3 Data Structures

A suitable data structure has to be set up that is capable of efficiently storing the information necessary for the optimization process. The efficiency is hereby determined by two properties. First of all, the required storage space. The less storage space is required to store the same information, the better. Secondly, whether often-user information can easily be accessed. Meaning, that if certain variables (metadata) are commonly asked for, a structure that stores these values will be preferred; this will prevent the algorithm from constantly having to recalculate these values. First, the MOLA problem that we build upon will be defined in more detail, including a data structure for storing a single solution. Afterwards, several data structures will be proposed that can be used to store additional metadata to either one solution or the archive (a set of solutions).

The problem will optimize O objectives. For each objective $o \in \mathcal{O}$, where $\mathcal{O} = \{1, ..., O\}$, an incremental objective function $f_o(s)$ will be available as was defined in Definition 6.2.1. The solutions will be subject to C constraints, of which each constraint $c \in \mathcal{C}$ (with $\mathcal{C} = \{1, ..., C\}$) satisfies Definition 6.2.2. A solution s exists out of a two dimensional array, representing a grid of width X and height Y. As such, $s[p]$ contains the land-use type of cell $p = (x = \{1, ..., X\}, y = \{1, ..., Y\})$. For each land-use type $t \in \mathcal{T}$, where $\mathcal{T} = \{1, ..., T\}$, a variable static($t$) is defined that states whether or not this type is static or not. Each solution s can be identified using a unique id(s) and has a valid(s) and explored(s) property that can be used to indicate whether a solution is valid, respectively explored. These are the minimal solution variables necessary to run PLS for MOLA and are always assumed to be present.

In case certain solution information needs to be calculated often, metadata can be added to keep track of this information. Storing metadata can be beneficial in case incrementally updating this information is less expensive than constantly recalculating these values from scratch. We will now define two metadata structures that can be added to a solution s and one that can be added to a set of solutions, called an archive $\mathcal{A}$. The first two metadata structures (MD-I and MD-II) are considered as 'essential', as these are able to reduce the computational complexity of PLS in all cases. This is caused by a (large) reduction in the number of computations necessary to update the archive, as will be shown in Chapter 6.12. The final metadata structures (MD-III) is considered as 'optional', as this structure is especially useful when

specific objectives are being optimized or constraints are validated. This will be shown in Chapter 6.8 and 6.13.

Definition 6.2.3 (MD-I)**. Essential:** A metadata structure that concerns the addition of an array $f(s)$ with $|f(s)| = O$ to each solution $s \in \mathcal{A}$. The array stores the objective value $f_o(s)$ of s at position o for every objective $o = \{1, ..., O\}$.

Definition 6.2.4 (MD-II)**. Essential:** A metadata structure that concerns the addition of a two-dimensional array $v(\mathcal{A})$ with $|v(\mathcal{A})| = O$ to the archive $\mathcal{A}$. At position o for every objective $o = \{1, ..., O\}$, the array stores an array of size $|\mathcal{A}|$ with the $id(s)$ of each solution $s \in \mathcal{A}$ sorted by increasing objective value $f_o(s)$.

Definition 6.2.5 (MD-III)**. Optional:** A metadata structure that concerns the addition of an array $n(s)$ with $|n(s)| = T$ to each solution $s \in \mathcal{A}$. The array stores the total number of cells with land-use type t of s at position t for every land-use type $t = \{1, ..., T\}$.

The first metadata structure proposed, called MD-I, is shown in Definition 6.2.3. It concerns the addition of an array to every solution that contains the objective value of that solution for every objective. The second metadata structure, MD-II, is shown in Definition 6.2.4. This structure is added to the archive (set of solutions) of the PLS algorithm. It concerns a two-dimensional array with all solutions sorted per objective value for each objective. Finally, the third metadata structure, MD-III, is shown in Definition 6.2.5. This structure concerns the addition of an array to each solution, with the total number of cells of each land-use type. In case a metadata structure is used in an upcoming operator or algorithm, a reference will be made to the corresponding metadata code.

At the start of the PLS algorithm, all metadata values will need to be computed. This is done using the initialization procedure of Algorithm 14. The initialization algorithm only needs to be run once. Afterwards, the metadata values will be updated incrementally after a change has occurred to the solution. The metadata regarding the archive of PLS does not have to be initialized, as it does not contain any solutions at the start. The initialization procedures of the two solution related metadata structures are shown in Algorithm 31 and 33 of Appendix A. Finally, the solution is being validated using Algorithm 29. This procedures determines and stores whether the solution is currently obeying the constraints. More details on this procedure can be found in Chapter 6.13. After the initialization procedure has completed, all information necessary for the optimization process is available.

Algorithm 14 Initialize(s)

```
1: input: a solution s
2:
3: for all metadata MD-x used in s do
4:     s := InitializeMD-x(s)                    ▷ Appendix A
5: end for
6:
7: valid(s) := Validate(s)                       ▷ Chapter 6.13
8:
9: output: s
```

In future research, it might also be considered to add a list of all boundary cells to a solution as an extra metadata structure. This list can be used for search and repair operators that make use of boundary cells. However, this metadata structure is expensive, both performance and memory wise, to maintain. It would require the algorithm to constantly check all neighboring cells of any adjusted cell to see whether or not these are (still) boundary cells. Since this might not be worth the gain in performance for these specific operator (and would so require more research), the decision has been made to leave this structure out of the current research.

Finally, we need to define one last data structure for the changes that are being made to a solution. These changes can be used to incrementally update the metadata or calculate the new objective values of a solution. The changes will be stored and passed on using an update set, as defined in Definition 6.2.6. The update set contains all cells that will be changed of a solution, together with the new land-use type they will obtain. The usage of an update set to incrementally update metadata is discussed in more detail in Chapter 6.12. In order to incrementally update objective values, the objective functions can be called with both the current solution and the update set. Examples of incremental objective functions of O1 (Cost Minimization) and O2 (Compactness Maximization) are shown in Algorithm 35 and 36 of Appendix A.

Definition 6.2.6 (Update Set)**.** An update set $\mathcal{U}$ contains only the cells of a solution that should be updated including their new type, stored as type(c) for each $c \in \mathcal{U}$.

6.3 Algorithm Definitions

The standard PLS and IPLS algorithms are applicable to any multi-objective optimization problem. However, using domain specific knowledge these can be tailored to a specific problem, possibly enhancing the algorithm its efficiency. In this chapter, both the PLS and IPLS algorithms are being modified to fit the properties of MOLA. The resulting MOLA-specific algorithms, named PLS-MOLA and IPLS-MOLA, will now be outlined, explained and justified in more detail.

6.3.1 Algorithm I: PLS-MOLA

The PLS-MOLA algorithm is based upon the standard Pareto Local Search of Algorithm 4. The overall outline of the algorithm is shown in Algorithm 15. PLS-MOLA uncouples the key processes of the standard PLS algorithm, allowing for an easy incorporation of different procedures. This includes for instance an uncoupling of the repair and search procedure, together with the repair and local search operators. Furthermore, the incorporation of two extra termination criteria allow for an easy configuration of the repair and neighborhood exploration strategies. The PLS-MOLA algorithm can therefore be considered as a configurable framework for the MOLA optimization process, that still retains the essence of PLS. The actual knowledge-informed procedures and/or operators for MOLA that can be implemented using this algorithm, are discussed in Chapter 6.4 to 6.14. The PLS-MOLA algorithm works as follows;

Line 1 - 8 The algorithm has seven parameters. The first one is the initial solution s_0 with the current land-use allocation. Next up, we have a set of repair operators $\mathcal{R}$, a set of local search operators $\mathcal{O}$ and an archive acceptance criterion $\mathcal{P}$. Finally, termination criteria T_1, T_2 and T_3 are required for the reparation procedure, neighborhood exploration and the algorithm itself respectively.

Line 9 - 20 The first step of the algorithm is to initialize all metadata structures that are being used. This is done once with the initialization procedure, as was discussed earlier in Chapter 6.2.3. Next up, the original solution will be repaired if it does not satisfy the constraints of the MOLA problem. This is done using a repair procedure that applies the required repair operators. Repairing will be discussed in more detail in Chapter 6.9. In case the repair procedure is unable to repair the original solution, PLS-MOLA will terminate and output an empty set. If the repair procedure succeeds or the solution was already valid, s_0 is set to unexplored and added to archive A_0 (set of all unexplored solutions) and A (set of all non-dominated solutions currently found).

Line 21 - 38 The next step is to continuously run the neighborhood exploration procedure. This procedure works as follows: first, a selection procedure is called to return an unexplored solution s from the archive A_0. Two different selection procedures will be examined in this research, both described in Chapter 6.4. Next up, a search procedure is called to return a neighborhood solution s' of $\mathcal{N}(s)$. The general search procedure being used in this research is outlined in Chapter 6.7, and several possible operators are proposed in Chapter 6.8. Possibly, s' needs to be repaired again using the repair procedure. The search and repair procedures are repeated until a valid solution s' is obtained. When such a valid solution is found, a check will be performed to see whether solution s' satisfies the acceptance criteria for being added to the archive. Three different acceptance criteria are described later on in Chapter 6.11. If accepted, s' is set to unexplored and A' (the temporary archive) will be updated with solution s'. The updating of an archive is discussed in more detail in Chapter 6.12. The neighborhood exploration procedure ends when termination criterion T_2 is met. An example of such a criterion can be a maximum number of neighborhood solutions that has been explored, as will be handled in Chapter 6.6. Finally, s is set to explored, archive A is updated with A', and A_0 will be filled with the currently unexplored solutions. As the metadata is updated within the search and repair operators, no further update function needs to be called.

Line 39 - 41 The algorithm will continuously perform a new neighborhood exploration, until eventually a termination criterion T_3 is met. This research will implement the termination criterion of Definition 6.3.1, named T-PLS$_{nm}$. The criterion ends the optimization process after either a total of n iterations have been done or the archive $\mathcal{A}$ has remained unchanged for m consecutive iterations. When the PLS-MOLA algorithm terminates, archive $\mathcal{A}$ will be the final output.

Definition 6.3.1 (T-PLS$_{nm}$)**.** A termination criterion that terminates the PLS-MOLA optimization loop after a total of n neighborhood explorations have been completed or the archive has remained unchanged for the last m neighborhood explorations.

Algorithm 15 PLS-MOLA(s_0, $\mathcal{R}$, $\mathcal{O}$, $\mathcal{P}$, T_1, T_2, T_3)

input: solution s_0 with the current land-use allocation
set $\mathcal{R}$ with repair operators
set $\mathcal{O}$ with local search operators
archive acceptance criterion $\mathcal{P}$
termination criterion T_1 for the reparation procedure
termination criterion T_2 for the neighborhood exploration
termination criterion T_3 for the PLS-MOLA algorithm

s_0 := Initialize(s_0) ▷ Chapter 6.2
if !valid(s_0) **then**
  s_0 := Repair(s_0, $\mathcal{R}$, T_1) ▷ Chapter 6.9
  if !valid(s_0) **then**
    output: $\emptyset$
  end if
end if

explored(s_0) := $false$
$\mathcal{A}_0 := \{s_0\}$
$\mathcal{A} := \mathcal{A}_0$

repeat
  $\mathcal{A}' := \mathcal{A}$
  s := Select($\mathcal{A}_0$) ▷ Chapter 6.4
  repeat
    repeat
      s' := Search(s, $\mathcal{O}$) ▷ Chapter 6.7
      if !valid(s') **then**
        s' := Repair(s', $\mathcal{R}$, T_1) ▷ Chapter 6.9
      end if
    until valid(s')
    if s' satisfies acceptance criteria $\mathcal{P}$ **then** ▷ Chapter 6.11
      explored(s') := $false$
      $\mathcal{A}'$:= UpdateA($\mathcal{A}'$, {s'}) ▷ Chapter 6.12
    end if
  until stopping criterion T_2 is met ▷ Chapter 6.6
  explored(s) := $true$
  $\mathcal{A}$:= UpdateA($\mathcal{A}$, $\mathcal{A}'$)
  $\mathcal{A}_0 := \{s \in \mathcal{A} \mid \text{explored}(s) = false\}$
until termination criterion T_3 is met

output: $\mathcal{A}$

6.3.2 Algorithm II: IPLS-MOLA

The IPLS-MOLA algorithm is based upon the Iterated Pareto Local Search algorithm that was defined in Algorithm 8. The overall outline of the algorithm is shown in Algorithm 16. Similar to PLS-MOLA, the algorithm tends to decouple different components to allow for an easy configuration. New in comparison to PLS-MOLA, is the use of a perturbation procedure to perturb a solution from the archive.

Line 1 - 10 The algorithm has nine different parameters. The first seven parameters are similar to the parameters of PLS-MOLA. In addition, a termination criterion for the perturbation procedure and IPLS-MOLA are added.

Line 11 - 12 At first, PLS-MOLA is run once to gain a starting archive. This can be done by simply passing on the first seven parameters. Note, that there is no need for a reparation procedure of s_0, as this is already incorporated in PLS-MOLA.

Line 13 - 22 Next up, multiple iterations are done to improve upon the archive. A single iteration starts off by selecting a solution s from the archive $\mathcal{A}$ using a selection procedure from Chapter 6.4. Next up, a perturbation procedure is started using s to create s'. The perturbation procedure and its termination criterion will be discussed in more detail in Chapter 6.14. Possibly, solution s' needs to be repaired using the repair procedure. The process of perturbing and repairing is repeated until a valid solution s' is found. Next up, PLS-MOLA is run again using solution s' as the initial solution. The remaining six parameters stay the same. Finally, the archive A is updated with the results. Optionally, the repairing of a perturbed solution can be outsourced completely to the PLS-MOLA algorithm (similar to the initial solution). However, by repairing beforehand the perturbation process and PLS-MOLA become more separated, allowing for instance termination criteria T_5 to target only IPLS operations more easily.

Line 23 - 25 The algorithm keeps on iterating until termination criterion T_5 is met. The termination criterion $\mathcal{A}_5$ for IPLS-MOLA can be similar to the termination criterion used for PLS-MOLA (T-PLS$_{nm}$) in Definition 6.3.1. The optimization process then ends after a total of n iterations have been done or the archive $\mathcal{A}$ has remained unchanged for m consecutive iterations. When the IPLS-MOLA algorithm terminates, the archive $\mathcal{A}$ will be the final output.

Algorithm 16 IPLS-MOLA(s_0, $\mathcal{R}$, $\mathcal{O}$, $\mathcal{P}$, T_1, T_2, T_3, T_4, T_5)

input: solution s_0 with the current land-use allocation
set $\mathcal{R}$ with repair operators
set $\mathcal{O}$ with local search operators
archive acceptance criterion $\mathcal{P}$
termination criterion T_1 for the reparation procedure
termination criterion T_2 for the neighborhood exploration
termination criterion T_3 for the PLS-MOLA algorithm
termination criterion T_4 for the perturbation procedure
termination criterion T_5 for the IPLS-MOLA algorithm

$\mathcal{A}$:= PLS-MOLA(s_0, $\mathcal{R}$, $\mathcal{O}$, $\mathcal{P}$, T_1, T_2, T_3) ▷ Algorithm 15

repeat
  s := Select($\mathcal{A}$) ▷ Chapter 6.4
  repeat
    s' := Perturb(s, $\mathcal{O}$, T_4) ▷ Chapter 6.14
    if !valid(s') **then**
      s' := Repair(s', $\mathcal{R}$, T_1) ▷ Chapter 6.9
    end if
  until valid(s')
  $\mathcal{A}'$:= PLS-MOLA(s', $\mathcal{R}$, $\mathcal{O}$, $\mathcal{P}$, T_1, T_2, T_3)
  $\mathcal{A}$:= UpdateA($\mathcal{A}$, $\mathcal{A}'$) ▷ Chapter 6.12
until stopping criterion T_5 is not met

output: $\mathcal{A}$

6.4 Selection Procedure

The first step in every iteration of the optimization procedure of PLS-MOLA, is selecting the next solution that will be explored. The selection procedure is responsible for selecting this solution from the archive, using a selection operator. In this research, one simple selection procedure is used that randomly selects a selection operator out of all selection operators passed on to the algorithm. The procedure is shown in Algorithm 17. In future research, the selection procedure could be expanded to choose the next selection operator being applied based on for example situations or results obtained from previous selections (Chapter 11).

Algorithm 17 Select($\mathcal{A}$, $\mathcal{O}$)

input: an archive $\mathcal{A}$
  a set $\mathcal{O}$ with selection operators ▷ Chapter 6.8

s := apply random selection operator from $\mathcal{O}$ to $\mathcal{A}$

output: s

6.5 Selection Operators

Different selection operators can be implemented to select a solution from the current archive. In this research, two different operators are defined; one based on randomness (S-R) and one based on the crowding distance (S-CD).

6.5.1 Operator I: S-R

The first selection operator is also used in the standard PLS algorithm of Algorithm 4. It simply selects a solution at random from the archive $\mathcal{A}$. The operator is straightforward, as shown in Algorithm 18.

Algorithm 18 S-R($\mathcal{A}$)

```
1: input: an archive A
2:
3: s := select random solution s from A
4:
5: output: s
```

6.5.2 Operator II: S-CD

The second selection operator makes use of the crowding distance (CD) property. For more details on the CD property, view Chapter 4.3.5. A higher crowding distance represents a 'less crowded' solution and is therefore preferred. As such, the selection operator simply returns the solution from the archive with the highest crowding distance value. The operator is similar to the optimistic hypervolume strategy of Chapter 5.3.1, but uses the crowding distance property instead of the OHI value (Equation (5.1)). The advantage of using the crowding distance, is that it scales more easily to more than two objectives. Algorithm 19 shows the resulting selection operator S-CD.

Algorithm 19 S-CD($\mathcal{A}$)

```
1: input: an archive A
2:
3: s := ∅
4: cd := 0
5: for all s' ∈ A do
6:     cd' = CD-PLS(s', A)                    ▷ Algorithm 38
7:     if cd' > cd then
8:         s := s'
9:         cd := cd'
10:    end if
11: end for
12:
13: output: s
```

Calculating the crowding distance of each solution of the archive can be a computationally expensive task. As such, a CD algorithm has been defined that makes use of MD-II, which concerned the addition of an array to the archive with all solutions sorted per objective value. Using this array the crowding distance can be calculated more efficiently. The algorithm, named CD-PLS, is shown in Algorithm 38 of Appendix D. In contrast to the original

crowding-distance calculation, the crowding distance of an extreme solution is now two times the distance to its nearest solution. This is similar to OHI in Equation (5.1), and prevents the operators from always selecting the extremes (as these would have infinite CD values). When implementing the S-CD selection operator, usage of the MD-II structure can be considered as essential. Calculating the crowding distance of all solutions from the archive from scratch is an expensive task, whereas incrementally updating the MD-II structure requires (much) less operations.

6.6 Neighborhood Exploration

Once a solution has been selected from the archive, different neighborhood exploration approaches can be applied for searching its neighborhood. In Chapter 5.3.3, several different strategies were discussed. The exploration strategy is passed on to the PLS-MOLA or IPLS-MOLA algorithm using a termination criterion T, as was shown in Algorithm 4 and 8. In this research, the termination criterion T-NE$_{nm}$ is used, which is defined in Definition 6.6.1. The termination criterion T-NE$_{nm}$ ends the neighborhood exploring loop after either n neighborhood solutions have been explored or m neighborhood solutions have been added to the archive. In the following subchapter, two exploration strategies will be discussed that can be implemented using specific settings of the T-NE$_{nm}$ criterion.

Definition 6.6.1 (T-NE$_{nm}$)**.** A termination criterion that terminates the neighborhood exploration process after either n neighborhood solutions have been explored or m neighborhood solutions have been added to the archive.

6.6.1 Strategy I: T-NE-BPI

The first exploration strategy that will be defined is the Best Pareto Improvement (BPI) strategy of Algorithm 11. The BPI strategy tends to explore the complete neighborhood of a solution, assuring that the best solution(s) present in this neighborhood will always be added to the archive. Theoretically, this strategy can be defined as T-NE$_{nm}$ with both $n = |\mathcal{N}(s)|$ and $m = |\mathcal{N}(s)|$, where $\mathcal{N}(s)$ contains all solutions from the neighborhood of solution s that is being explored. However, the neighborhood of a solution in MOLA can be incredibly large, which would (when applied to a realistic case) result in extremely long exploration times. As such, we do not determine the complete neighborhood before exploring its solutions, and sample solutions from the neighborhood space. This will be discussed in more detail in Chapter 6.7, where we will elaborate on the search procedure. As the complete neighborhood is not defined before the exploration, the total size of the neighborhood is also unknown. Therefore, the BPI strategy will be implemented (approached) by using T-NE$_{nm}$ with both $n = c$ and $m = c$ where c is some large integer. The resulting T-NE-BPI$_c$ strategy is defined in Definition 6.6.2. A larger c value will result in a more close approximation of the BPI strategy, at the cost of more processing time.

Definition 6.6.2 (T-NE-BPI$_c$)**.** The termination criterion of T-NE$_{nm}$ with $n = c$ and $m = c$ where $c >> 1$.

6.6.2 Strategy II: T-NE-FPI

The second exploration strategy that will be defined using T-NE$_{nm}$, is the First Pareto Improvement (FPI) strategy of Algorithm 13. The FPI strategy tends to explore the neighborhood until a single new solution is accepted and added to the archive. Theoretically, this strategy can be implemented using T-NE$_{nm}$ with $n = |\mathcal{N}(s)|$ and $m = 1$, where $\mathcal{N}(s)$ contains all solutions from the neighborhood of solution s that is being explored. However, as mentioned before the neighborhood is not defined before the exploration and so its size is unknown. As such, n is set to a large value c, similar to T-NE-BPI$_c$. The resulting T-N-FPI$_c$ strategy is defined in Definition 6.6.3. The larger c, the closer the T-NE-FPI$_c$ criterion will implement the FPI strategy, but the more processing time the exploration of a solution its neighborhood might cost.

Definition 6.6.3 (T-NE-FPI$_c$)**.** The termination criterion of T-NE$_{nm}$ with $n = c$ and $m = 1$ where $c >> 1$.

6.7 Search Procedure

The search procedure is responsible for returning a solution from another solution (the selected one) its neighborhood. The procedure does so, by applying a local search operator to the current solution being explored. In this research, only one search procedure will be used, that simply applies a randomly selected search operator from the set of search operators passed on to the algorithm. The procedure is shown in Algorithm 6.8. As already mentioned in Chapter 6.6, the complete neighborhood of a solution in a (realistic) MOLA case can be incredibly large. As so, it can be very costly to determine the complete neighborhood of a solution in advance and then sample from or iterate over this set. The current search procedure therefore randomly samples solutions directly from the neighborhood space, until the desired number of solutions from the exploration termination criterion have been explored or added to the archive. This means that possibly a neighborhood solution might be sampled twice during one neighborhood exploration. However, due to the in general significant size of a neighborhood, the number of twice sampled solutions will most likely be relatively low. As such, it is considered unnecessary to keep track of all solutions already returned (which can be very costly). In future research, other search procedures can be implemented that for example select operators with a certain predefined or automatically adapting chance (Chapter 11). For now, this was out of the scope of this research.

Algorithm 20 Search(s, $\mathcal{O}$)

1: **input:** a solution s
2: a set $\mathcal{O}$ with local search operators ▷ Chapter 6.8
3:
4: s := apply random local search operator from $\mathcal{O}$ to s
5:
6: **output:** s

6.8 Search Operators

Multiple local search operators can be passed on to the PLS-MOLA or IPLS-MOLA algorithm to explore the neighborhood of a solution. As shown in Algorithm 6.7, each time the search procedure is called, a random local search operator is chosen from this set and applied to the current solution being explored. In this section, three local search operators for MOLA will be proposed. The first two operators are independent of the objectives being optimized, whereas the third operator is designed to stimulate compactness maximization.

6.8.1 Operator I: LS-KRCM

The first local search operator for PLS-MOLA that will be defined is the k-tournament random cell mutation (LS-KRCM) operator. The operator is based upon the BRCM mutation operator discussed in Chapter 4.3.7. It changes the land-use of one random cell to a type that is decided on using an objective value based tournament. The pseudo-code of the operator is shown in Algorithm 21.

The LS-KRCM operator works as follows. It is called with two parameters: the solution s to which the operator is applied, and an integer k_t for the size of the type selection tournament. The algorithm starts off by selecting a random non-static cell c from s, to which a new land-use type will be allocated. The selection of this new land-use type is based upon a randomly selected objective o and works as follows. First, k_t different land-use types are randomly selected from the set of all dynamic land-use types (excluding the current type of c). In case k_t is equal to the total number of dynamic land-use types, the complete set is taken. The incremental value change in the objective that was selected earlier, is calculated for each of the selected types. This is done by passing on this change (as an update set $\mathcal{U}$ with the cell and its new type) to the incremental objective function, as was discussed in Chapter 6.2.3 (Definition 6.2.6). The incremental objective function of O1 (Cost Minimization) and O2 (Compactness Maximization) can be found in Appendix B. Since we demand all objectives to be maximized, the type that results in the highest (most positive) incremental value change will be chosen. The update set stating that this type should be allocated to the selected cell, will be passed on to a separate update function that will apply this change to s and update its metadata accordingly. Solution updating will be discussed in more detail in Chapter 6.12. By separating this process from the search operator, searching and updating become loosely coupled and can be adjusted separately. After solution s has been updated, it is returned as the final output of the LS-KRCM operator.

Algorithm 21 LS-KRCM(s, k_t)

input: a solution s
        an integer k_t for the tournament size

c := randomly select a non-static cell c from s
o := randomly select an objective $o \in \mathcal{O}$

$\mathcal{T}' := \{t \in \mathcal{T} \mid \text{!static}(t)\} \setminus \text{type}(c)$
if $|\mathcal{T}'| > k_t$ **then**
    $\mathcal{T}'$:= randomly take k_t types from $\mathcal{T}'$
end if

$\mathcal{U}_{best} := \emptyset$
for all land-use type $t \in \mathcal{T}'$ **do**
    $\mathcal{U} := \{c\}$ with type(c) := t
    if $f_o(s, \mathcal{U}) > f_o(s, \mathcal{U}_{best})$ **then** ▷ Appendix B
        $\mathcal{U}_{best} = \mathcal{U}$
    end if
end for

s := UpdateS(s, $\mathcal{U}_{best}$) ▷ Chapter 6.12

output: s

6.8.2 Operator II: LS-KRPM

The second local search operator is the k-tournament random patch mutation (LS-KRPM) operator. The operator is shown in Algorithm 22, and is based upon the RPM mutation operator of Chapter 4.3.7. Instead of a single cell, the operator mutates a complete 7-cell patch. First, a patch figure is generated by randomly selecting 7 cells out of a 3 x 3 grid, which always results in one continuous patch (to stimulate compactness). Next, a random valid location is selected for the patch, that does not contain any static land-use types. The operator will now allocate one single land-use type to all 7 cells of the patch (denoted as the set $\mathcal{P}$), which is selected using a tournament in the same manner as in LS-KRCM. However, instead of calculating the change in objective value when allocating a type to one cell, it will now calculate the change for when a type is allocated to a complete patch. Finally, the solution is updated given all cells of the patch, accompanied with the newly selected type. The resulting solution is provided as the final output of the operator.

Algorithm 22 LS-KRPM(s, k_t)

input: a solution s
        an integer k_t for the tournament size

$\mathcal{P}$:= randomly select a 7-cell patch with no static cells ▷ Chapter 4.3.7
o := randomly select an objective $o \in \mathcal{O}$

$\mathcal{T}' := \{t \in \mathcal{T} \mid \text{!static}(t)\}$
if $|\mathcal{T}'| > k_t$ **then**
    $\mathcal{T}'$:= randomly take k_t types from $\mathcal{T}'$
end if

$\mathcal{U}_{best} := \emptyset$
for all land-use type $t \in \mathcal{T}'$ **do**
    $\mathcal{U} := \mathcal{P}$ with type(c) := t for all $c \in \mathcal{P}$
    if $f_o(s, \mathcal{U}) > f_o(s, \mathcal{U}_{best})$ **then** ▷ Appendix B
        $\mathcal{U}_{best} = \mathcal{U}$
    end if
end for

s = UpdateS(s, $\mathcal{U}_{best}$) ▷ Chapter 6.12

output: s

6.8.3 Operator III: LS-KRBM

The third and final local search operator being proposed, is the k-tournament random boundary cell mutation (LS-KRBM) operator. Whereas the LS-KRCM and LS-KRPM did not focus on any particular objective, the LS-KRBM is designed to specifically promote compactness maximization. In case this objective is not maximized, usage of the LS-KRBM will not provide any advantages. The procedure of LS-KRBM is shown in Algorithm 23 and mutates the land-use type of a boundary cell. The operator works similar to LS-KRCM, except for two major differences. The first difference, is that the randomly selected cell is a boundary cell. This cell will be selected by either storing the location of each boundary cell or sampling random cells until a boundary cell is found. The second difference is that its new land-use type is only selected out of the types of its non-static neighboring solutions. Both of these differences serve the same goal; to reduce the chance of creating new clusters, which promotes the compactness maximization objective. Apart from these differences, the LS-KRBM operator works similar to LS-KRCM.

Algorithm 23 LS-KRBM(s, k_t)

input: a solution s with:
        an integer k_t for the tournament size

bc := randomly select a non-static boundary cell bc from s
o := randomly select an objective $o \in \mathcal{O}$

$\mathcal{T}' := \{t \in \mathcal{T}_{\mathcal{N}(bc)} \mid \text{!static}(t)\} \setminus \text{type}(bc)$
if $|\mathcal{T}'| > k_t$ **then**
    $\mathcal{T}'$:= randomly take k_t types from $\mathcal{T}'$
end if

$\mathcal{U}_{best} := \emptyset$
for all land-use type $t \in \mathcal{T}'$ **do**
    $\mathcal{U} := \{bc\}$ with type(bc) := t
    if $f_o(s, \mathcal{U}) > f_o(s, \mathcal{U}_{best})$ **then** ▷ Appendix B
        $\mathcal{U}_{best} = \mathcal{U}$
    end if
end for

s := UpdateS(s, $\mathcal{U}_{best}$) ▷ Chapter 6.12

output: s

6.9 Reparation Procedure

In order to repair an invalid solution that violates the problem constraints, the reparation procedure will be used. A simple reparation procedure that will be used throughout this research, is shown in Algorithm 24. Until the termination criterion T is met, the algorithm will attempt to repair the solution for as long as one of the constraints is violated. It is assumed (and required) that for each constraint c, a validation method is available that states whether or not a solution satisfies this constraint. An example of such a validation method for constraint C1 is shown in Algorithm 37 of Appendix E. Note, that this validation method makes use of the metadata structure (MD-III) that stores the total number of cells of each land-use type, to easily detect a violation of the allowed range. Validation methods are not passed on as a parameter, as they are problem specific and assumed to always be present (similar to objective functions). In case a constraint is violated, the corresponding repair operator from $\mathcal{R}$ will be applied to (try to) fix this violation. The repair operator set $\mathcal{R}$ should therefore contain at least one repair operator for each constraint. Optionally, these repair operators can again require specific termination criteria, which we for now assume not to be present. As the metadata of the solution is being updated within the repair operator (using the update function), constraint validation using metadata can continue to be done in the reparation loop. This process continues until no more constraints are violated or the termination criterion is met. In Chapter 6.10, two repair operators have been defined for the most common C1 constraint.

Algorithm 24 Repair(s, $\mathcal{R}$, T)

```
1: input: solution s to be repaired
2:        set R with repair operators                          ▷ Chapter 6.10
3:        termination criterion T_1
4:
5: repeat
6:     for all constraint c = {1, ..., C} do
7:         if !Validate-c(s) then                               ▷ Appendix C
8:             s := apply repair operator for c from R to s     ▷ Chapter 6.10
9:         end if
10:    end for
11: until valid(s) or termination criterion T is met
12:
13: output: s
```

In this research, a reparation termination criterion called T-REP$_{nm}$ will be used. The definition of T-REP$_{nm}$ is shown in Definition 6.9.1. The n hereby stands for the maximum number of iterations in the reparation procedure and m for the maximum number of consecutive repair operations for which the solution may remain unchanged. A solution can remain unchanged in case the operator fails to find an improvement (view for example the LR-KBRM operator in Chapter 6.10.2).

Definition 6.9.1 (T-REP$_{nm}$)**.** A termination criterion that terminates the reparation procedure loop in case a repair operator has been applied to the solution n times or the solution has remained unchanged during the last m repair operators applied.

6.10 Repair Operators

A repair operator needs to be defined for each constraint that can be violated. As discussed in Chapter 6.2, this research will only provide components and operators for the allocation range constraint (C1). This has resulted in the definition of two repair operators: LR-KCRM and LR-KBRM.

6.10.1 Repair Operator: LR-KCRM

The first operator that will be defined is the K-Tournament Cell Repair Mutation (LR-KCRM) operator that aims to repair a solution that violates constraint C1. The operator is run with two input parameters; a solution s that is invalid in terms of the C1 constraint and an integer k_c with the size of the cell selection tournament. The solution s needs to include one metadata structure; an array $n(s)$ with the number of cells per land-use type (MD-III). The repair operator is depicted in Algorithm 25. The operator first detects of which land-use types the number of cells are currently below the lower bound ($\mathcal{T}_{lb}$), above the upper bound ($\mathcal{T}_{lb}$) and in between ($\mathcal{T}_b$). This can be done efficiently using MD-III, which contains to how many cells each type is allocated. Next, a tournament between k_c cells is held, which works as follows. For each of the k_c iterations, a random cell will be chosen. In case one or more types violate the upperbound, this cell needs to be of one of these types. Otherwise, it needs to be of a type that violates the lower bound. If not, a new cell will be selected until this holds. After a cell has been found to repair, the

new type of this cell will be selected. In case the lower bound is violated, this new type needs to be of a type that does so. Otherwise, the new type needs to be a type that is already in between the bounds. For all of these types, the potential change in objective value of a randomly selected objective that was chosen at the beginning, will be calculated. Eventually, the cell and type that result in the best objective value improvement, will be saved and used to update solution s. The repair operator guaranties to reduce the 'amount' of violation of C1 after being applied. The operator hereby assumes that at least one type of the given solution violates constraint C1. With the cell selection tournament size, the user is able to influence the duration (costs) and quality (regarding the objective values) of the repaired solution. Especially at the start of PLS-MOLA, the initial solution might need a large number of reparation operations to become valid. The higher k_c, the longer the reparation will take, but the better the eventual objective values are expected to be.

Algorithm 25 LR-KCRM(s, k_c)

input: solution s with:
  an array $n(s)$ with the number of cells per type ▷ MD-III
  an integer k_c with the cell tournament size

o := randomly select an objective $o \in \mathcal{O}$
$\mathcal{T}_{lb} := \{t \in \mathcal{T}' | n(s)[t] < LB_t\}$
$\mathcal{T}_{ub} := \{t \in \mathcal{T}' | n(s)[t] > UB_t\}$
$\mathcal{T}_b := \mathcal{T} \setminus (\mathcal{T}_{lb} \cup \mathcal{T}_{ub})$

$\mathcal{U}_{best} := \emptyset$
for k_c times **do**
  c := randomly select a non-static cell c from s
  if $|\mathcal{T}_{ub}| > 0$ **then**
    while type(c) $\notin \mathcal{T}_{ub}$ **do**
      c := randomly select a non-static cell c from s
    end while
  else
    while type(c) $\notin \mathcal{T}_{lb}$ **do**
      c := randomly select a non-static cell c from s
    end while
  end if

  $\mathcal{T}' := \mathcal{T}_b$
  if $|\mathcal{T}_{lb}| > 0$ **then**
    $\mathcal{T}' := \mathcal{T}_{lb}$
  end if

  for all land-use type $t \in \mathcal{T}'$ **do**
    $\mathcal{U}$:= {c} with type(c) := t
    if $f_o(s, \mathcal{U}) > f_o(s, \mathcal{U}_{best})$ **then** ▷ Appendix B
      $\mathcal{U}_{best} = \mathcal{U}$
    end if
  end for
end for

s := UpdateS(s, $\mathcal{U}_{best}$) ▷ Chapter 6.12

output: s

6.10.2 Repair Operator: LR-KBRM

The K-Tournament Boundary Cell Repair Mutation (LR-KBRM) is the second repair operator that can be used to repair a solution that violates constraint C1. Again, the operator is run given a solution s that violates the C1 constraint and an integer k_c that states the size of the cell selection tournament. The repair operator is similar to the LR-KCRM operator, except for two major differences. First of all, only boundary cells are mutated. Similar to LS-KBRM, this can either be done by taking note of the boundary cells of a solution or randomly sampling cells until a boundary cell is obtained. The operator keeps on sampling boundary cells, until a boundary cell bc with a type that violates the C1 constraint is found. The second difference, is the set

of optional types that can be allocated to this cell; only types of neighboring cells are allowed. Now lets consider the first case, in which the boundary cell violates the lower bound. In that case a tournament will be performed between the types of all neighboring cell that do not violate the lower bound either, to decide which one is best for the *bc* cell. In this case, the best type is the one that results in the highest positive change regarding the randomly selected objective at the start (assuming we are maximizing). If this update would result in the best update so far, it is stored. In the second case, the type of the boundary cell violates the upper bound. This will result in a tournament between the neighboring types of the boundary cell that do not violate the upper bound as well, deciding which type will be allocated to *bc*. Again, in case this update would result in the best objective change found so far, it will be stored. This process will be repeated for k_c cells, of which eventually the best update found will be applied and the resulting solution will be returned. In contrast to LR-KCRM, the LR-KBRM does not assure that the resulting solution violates constraint C1 'less' than before; it is for example possible that a *bc* that violated the upper bound, only has neighbors that do so to. In that case, no change will be made. Remember, that the variable m is passed on to the reparation procedure, stating the maximum number of consecutive times that this is allowed to happen. Similar to the LS-KBRM operator, the LR-KBRM operator is made specifically to promote compactness. In case compactness maximization is not among the objectives being optimized, the operator has no advantage above the LR-KCRM operator; it even has a disadvantage due to fact that a random boundary cell is more expensive to obtain than a random cell. Similar to LR-KCRM, a higher k_c will result in a longer and more computationally expensive reparation process, but is also expected to result in higher objective values.

Algorithm 26 LR-KBRM(s, k_c)

input: solution s with:
  an array $n(s)$ with the number of cells per type ▷ MD-III
  an integer k_c with the cell tournament size

o := randomly select an objective $o \in \mathcal{O}$
$\mathcal{T}_{lb} := \{t \in \mathcal{T}' | n(s)[t] < LB_t\}$
$\mathcal{T}_{ub} := \{t \in \mathcal{T}' | n(s)[t] > UB_t\}$
$\mathcal{T}_b := \mathcal{T} \setminus (\mathcal{T}_{lb} \cup \mathcal{T}_{ub})$

$\mathcal{U}_{best} := \emptyset$
for k_c times **do**
  bc := randomly select a non-static boundary cell bc from s
  if $|\mathcal{T}_{ub}| > 0$ **then**
    while type(bc) $\notin (\mathcal{T}_{lb} \cup \mathcal{T}_{ub})$ **do**
      bc := randomly select a non-static boundary cell bc from s
    end while
  end if

  if type(bc) $\in \mathcal{T}_{lb}$ **then**
    $\mathcal{N}' := \{c \in \mathcal{N}(bc)\}$| !static(type($c$)) and type($c$) $\notin \mathcal{T}_{lb}\}$
    for all cell $c \in \mathcal{N}'$ **do**
      $\mathcal{U}$:= {c} with type(c) := type(bc)
      if $f_o(s, \mathcal{U}) > f_o(s, \mathcal{U}_{best})$ **then** ▷ Appendix B
        $\mathcal{U}_{best} = \mathcal{U}$
      end if
    end for
  else
    $\mathcal{T}' := \{t \in \mathcal{T}_{\mathcal{N}(bc)}|$!static(t) and $t \notin \mathcal{T}_{ub}\}$
    for all land-use type $t \in \mathcal{T}'$ **do**
      $\mathcal{U}$:= {bc} with type(bc) := t
      if $f_o(s, \mathcal{U}) > f_o(s, \mathcal{U}_{best})$ **then** ▷ Appendix B
        $\mathcal{U}_{best} = \mathcal{U}$
      end if
    end for
  end if
end for

s := UpdateS(s, $\mathcal{U}_{best}$) ▷ Chapter 6.12

output: s

6.11 Acceptance Criteria

Different acceptance criteria can be implemented to decide upon whether a newly found solution should be added to the archive or not. In this research, three different acceptance criteria will be discussed.

6.11.1 Criterion I: AC-ND

The first acceptance criterion is the most used criterion for PLS, which will be referred to as AC-ND. The criterion is defined in Definition 6.11.1. The AC-ND criterion accepts a solution s for inclusion in the archive $\mathcal{A}$ in case it is non-dominated by all solutions currently present in the archive. The AC-ND criterion is relatively weak and allows for an extensive exploration of the search space at the cost of a (possibly) quickly expanding archive.

Definition 6.11.1 (AC-ND)**.** A solution s is accepted for inclusion in $\mathcal{A}$ if $s' \not\prec s$ for all $s \in \mathcal{A}$.

6.11.2 Criterion II: AC-NDS

The second acceptance criterion is a stronger form of AC-ND, which will be referred to as AC-NDS. The criterion is defined in Definition 6.11.2. The AC-NDS criterion only accepts a solution s' from the neighborhood of solution s that is currently being explored, if it is not dominated by any solution currently in the archive $\mathcal{A}$ and s' dominates s. As this criterion is stronger (more strict) than AC-ND, it will result in a smaller archive. As such, AC-NDS could allow for a faster approximation of the Pareto front at the cost of a decreased diversity.

Definition 6.11.2 (AC-NDS)**.** A solution s' from the neighborhood of solution s that is being explored, is accepted for inclusion in $\mathcal{A}$ if $s' \prec s$ and $s'' \not\prec s'$ for all $s'' \in \mathcal{A}$.

6.11.3 Criterion III: AC-CD

The final acceptance criterion will be referred to as AC-CD$_n$ and is defined in Definition 6.11.3. When using AC-CD$_n$, a maximum archive size n is being maintained to prevent the archive from expanding too rapidly. In case the archive size $|\mathcal{A}|$ is below n, it maintains the AC-ND$_n$ criterion. After $|\mathcal{A}|$ reaches its maximum allowed size, a solution s will only replace a solution $s' \in \mathcal{A}$ if AC-ND$_n$ is met and the crowding distance of s is higher than the crowding distance of s'. As such, a high diversity is maintained among the select group of solutions in the archive. The lower the maximum size of the archive, the quicker PLS will convergence, but the lower the size of the final Pareto front is expected to be.

Definition 6.11.3 (AC-CD$_n$)**.** A solution s is accepted for inclusion in $\mathcal{A}$ if $s' \not\prec s$ for all $s \in \mathcal{A}$ and $|\mathcal{A}| < n$ or else replaces the solution $s'' \in \mathcal{A}$ with the smallest CD, if $s' \not\prec s$ for all $s \in \mathcal{A}$ and CD(s) > CD(s'').

Since all criterion require the solution to not be dominated by any solution in the archive, the AC-ND criterion has been implemented in the archive update function of Algorithm 28, that will be discussed Chapter 6.12. In order to check this criterion efficiently, the metadata structure MD-II is used which adds an array to the archive that keeps track of all solutions sorted per objective value. The AC-ND criterion is incorporated inside the update function, as certain information obtained during the validation of criterion AC-ND can efficiently be re-used for updating MD-II. The AC-NDS will also require a comparison between s and s' before updating the archive. Finally, the AC-CD criterion will require a comparison in crowding distance in the update

function of Algorithm 28, before adding a solution to the archive. This can efficiently be implemented using the CD function for MOLA of 38 in Appendix D.

6.12 Updating Procedures

The local search or repair operators decide upon the way a solution can be adjusted to create a new solution from its neighborhood. However, they do not apply these adjustments to the solution themselves; this is done using a separate updating procedure. The advantage of separating the operators and updating procedure, is that the algorithm becomes more loosely coupled. Where the aim of a search or repair operator is to decide upon the changes to create a new solution, the updating procedure only aims to update this solution and its metadata as efficiently as possible, given these changes. As so, an adjustment in for example the updating procedure does not require an adjustment in any of the search or repair operators. There are two different updating procedures; one for updating a solution and one for updating an archive. First, we will discuss the solution updating procedure and afterwards we will handle the archive updating procedure.

6.12.1 Solution Updating

The updating procedure for solutions is depicted in Algorithm 27. It is called given the solution s to be updated and an update set $\mathcal{U}$ with the changes that should be applied to s. An update set is simply the set of all cells of s that should obtain a different land-use type including these new types (Definition 6.2.6). Generally, the algorithm consists out of three steps. First, the metadata of solution s is being updated. For each metadata structure that is being used, an updating procedure should be available. The updating procedure is also called given s and $\mathcal{U}$, and should update the corresponding metadata incrementally using the changes that will be applied to s. In this research, two metadata structures for a solution were defined; MD-I and MD-III. The incremental updating procedures of these two structures are shown in Algorithm 32 and 34 of Appendix A. In the next phase of the algorithm, the changes of the update set are being applied to solution s. The types of the cells in s that are mentioned in $\mathcal{U}$, will be changed to their new types. The final step, is to run the validation function on the new solution. This procedure updates the valid(s) variable that states whether s satisfies the constraints or not and is further discussed in Chapter 6.13. After the metadata structures have been updated, the changes have been applied and the new solution has been validated, the solution is successfully updated and can be returned.

Algorithm 27 UpdateS($s, \mathcal{U}$)

input: a solution s
        a set $\mathcal{U}$ containing the updates for s ▷ Definition 6.2.6

for all metadata MD-x used in s **do**
    s := UpdateMD-x($s, \mathcal{U}$) ▷ Appendix A
end for

for all cell $c \in \mathcal{U}$ **do**
    type($s[c]$) := type(c)
end for

valid(s) := Validate(s) ▷ Chapter 6.13

output: s

6.12.2 Archive Updating

Finally, an archive updating procedure is defined in Algorithm 28. The archive updating procedure is called given the current archive and a new archive (set of solutions) to be added to this archive. In order to do so efficiently, it makes use of the two metadata structures of Chapter 6.2.3 that were considered as 'essential'; the MD-I structure that stores the current objective values of a solution, and the MD-II structure that stores (an id of) each solution in the archive sorted per objective value. The archive updating procedure has three goals. First of all, it needs to determine whether solutions of $\mathcal{A}'$ satisfy the AC-ND acceptance criteria (Definition 6.11.1). This criteria requires a solution not to be dominated by any other solution currently in the archive. As discussed in Chapter 6.11.1, this criteria is always present and can be checked inside the updating procedure. The second and third goal of the updating procedure, are the actual updating of the archive and the archive related metadata structure MD-II (in case the criteria were satisfied).

The procedure works as follows. Each solution $s \in \mathcal{A}'$ that might be added to the archive, is checked separately. First, its position in the sorted array $v(\mathcal{A})$ is determined efficiently using its array $f(s)$ with objective values. All predecessors of s in the list regarding the first objective are stored in $\mathcal{P}$ and all successors of s are stored in $\mathcal{S}$. Note, that the solutions in $\mathcal{P}$ now dominate s regarding the first objective, whereas the solutions in $\mathcal{P}$ are dominated. Next, we iterate over all remaining objectives and intersect (not unite) $\mathcal{P}$ with the new predecessors and $\mathcal{S}$ with the successors. After all objectives have been processed, we now end up with a list $\mathcal{P}$ with solutions that dominate s on all objectives, and a list $\mathcal{S}$ with solutions that are dominated by s for all objectives. In case $\mathcal{P}$ is not empty, and so s is dominated by one or more solutions from the archive, the solution is rejected and we continue on to the next solution. In case $\mathcal{P}$ is empty, s is not dominated by any solutions in the archive, and will be added to archive. While doing so, all solutions that were dominated by s will be removed from the archive, and the sorted array $v(\mathcal{A})$ (MD-II) will be updated. After all solutions from $\mathcal{A}'$ are processed, the updated archive $\mathcal{A}$ will be returned as the output.

Algorithm 28 UpdateA($\mathcal{A}$, $\mathcal{A}'$)

input: an archive $\mathcal{A}$ with:
  an array $f(s)$ for each $s \in \mathcal{A}$ with obj. values ▷ MD-I
  an array $v(\mathcal{A})$ with all $s \in \mathcal{A}$ sorted per obj. value ▷ MD-II
  an archive $\mathcal{A}'$ for updating $\mathcal{A}$

for all $s \in \mathcal{A}'$ **do**
  v' := insert s in $v'[o]$ for all objectives $o \in \mathcal{O}$ using array $f(s)$
  $\mathcal{P} := \{s' \in v'[1] \mid s' \prec s\}$
  $\mathcal{S} := \{s' \in v'[1] \mid s' \succ s\}$
  for all objective $o = \{2, ..., O\}$ **do**
    $\mathcal{P} := \mathcal{P} \cap \{s' \in v'[o] \mid s' \prec s\}$
    $\mathcal{S} := \mathcal{S} \cap \{s' \in v'[o] \mid s' \succ s\}$
  end for
  if $\mathcal{P} == \emptyset$ **then**
    $\mathcal{A} := (\mathcal{A} \cup s) \setminus \mathcal{S}$
    $v(\mathcal{A})$:= v' with all $s' \in \mathcal{S}$ removed
  end if
end for

output: $\mathcal{A}$

6.13 Validating Procedure

A new solution generated by the updating procedure might violate the constraints of the MOLA problem. Therefore, a check has to be performed to verify whether the solution is still valid. For each constraint that can be violated, a validation function needs to be implemented. In Appendix C, the constraint validation function of constraint C1 (allocation ranges) is outlined in Algorithm 37. The function makes use of metadata structure MD-I to efficiently check whether the number of cells per land-use type violate the allowed ranges. The general constraint checking function that calls all validation scripts is shown in Algorithm 29. Although this research only takes C1 in account, more constraints and so validation functions can be added to Algorithm 29 easily.

Algorithm 29 Validate(s)

input: a solution s to validate

for all constraint $c \in \mathcal{C}$ **do**
  if !Validate-c(s) **then** ▷ Appendix C
    output: $false$
  end if
end for

output: $true$

6.14 Perturbation Procedure

The IPLS-MOLA algorithm of Algorithm 16 uses a perturbation procedure to create an initial solution for the next iteration out of a solution from the current archive. The goal of the perturbation procedure is to kick the solution from the archive out of its current local optimum. As such, the changes applied to the solution should be large enough to do so, but not too large to lose all qualitative solution parts. Therefore, the perturbation procedure will need to be 'tuned' to the MOLA problem it is being applied to in order to work efficiently.

The perturbation procedure used in this research is shown in Algorithm 30. Given a solution s and a perturbation size p, the procedure randomizes p percent of all dynamic cells of s. In order to do so, one big update set is build with a random type assigned to p percent of all dynamic cells and passed on to the solution updating function to ensure that all metadata is being updated as well. The larger the perturbation size p, the more the resulting solution will deviate from the original solution s. The minimum perturbation size is 0, resulting in no perturbing at all, whereas the maximum perturbation size of 100 results in a completely randomized solution. Note, that a random type might result in the same type that a cell already had. As such, the actual percentage of cells of which the type has changed will be below (or equal to) the perturbation size.

Algorithm 30 Perturb(s, p)

input: solution s
perturbation size p

$\mathcal{P}$:= randomly select p percent of all dynamic cells
$\mathcal{U} := \emptyset$

for all cell $c \in \mathcal{P}$ **do**
  $\mathcal{U} := \mathcal{U} \cup c$ with a random type
end for

s := UpdateS($s, \mathcal{U}$) ▷ Chapter 6.12

output: s

Chapter 7

Implementation

7.1 Setup

In order to be able to optimize and examine the performance of the PLS algorithm for MOLA, a proof of concept has been build. Whereas the functionalities of the algorithm are outlined in Chapter 6, this chapter will elaborate on how these can be translated and integrated into software. The overall program has been designed with the aim to minimize the program complexity and amount of memory required to store and run the algorithm. Reducing the complexity will improve the scalability of the algorithm, whereas reducing the amount of memory required will prevent the algorithm from requiring (unrealistically) high amounts of memory when optimizing large MOLA problems. Increasing the scalability will have the highest priority, as this was one of the main challenges currently being faced in the MOLA domain (Chapter 1.2).

The implementation has been created using .NET Framework 4.8 (Microsoft, 2019b). This is the latest version of .NET Framework, released in 2019. No libraries were used for the core functionalities of the algorithms. The reason for not using common algorithm libraries is that these are programmed for combinatorial optimization problems in general. In this research we would like to optimize PLS specifically for MOLA problems, meaning that we want to exploit problem-specific properties. As these libraries do not give us the maximum flexibility to do so, the algorithms have been build independently. However, in case a general static functionality was required, libraries were preferred (to reduce both the amount of work and possibility of errors). An example of such a functionality, is the calculation of the hypervolume (Chapter 7.2.3). In future research, the algorithm could be implemented in more low level programming languages than C# such as C++, to further enhance the performance. For now, the goal of the POC is mainly to compare the performance of different operators, strategies and algorithms, so this was out of the current project its scope. The final software program will be handed over to the Energy and Resources group of Utrecht University and possibly be made public over time.

In the following chapters, the design choices considering the implementation will be explained. The focus will hereby lay on the data structures used and their relationships. Handling more specific implementation details, such as method specific optimizations, is out of the scope of this document. If interested, these can be viewed in the source code if made public. At first, the

problem and solution framework will be outlined in Chapter 7.2. These structures will allow an optimization algorithm to read and solve a MOLA problem. Secondly, the implementation of three of these algorithms will be outlined; PLS, IPLS and NSGA-II in Chapter 7.3.1, 7.3.2 and 7.3.3. All structures are explained and documented using UML class diagrams. Finally, there are several other supporting classes that provide general functionalities (for example for IO handling). These will also shortly be mentioned in Chapter 7.2.3 and are added to Appendix E.

7.2 Framework

In order to allow different algorithms to optimize a MOLA problem, a small framework has been designed. The framework allows for the storage of a MOLA problem and solutions in an efficient and convenient manner. The focus of the framework is on incremental objective handling and solution updating, which is used to decrease the required memory storage space. Also, the framework allows for a more fair comparison between different algorithms, as all algorithm are able to access the MOLA problem in the same manner. At first, the problem structure will be discussed, followed by the solution structure. Finally, several small helping structures will be summarized. In future research, the framework can be reused when proposing and comparing improvements for PLS-MOLA or other MOLA-specific algorithms.

7.2.1 Problem Setup

The problem structure allows an algorithm to efficiently access and interact with the objectives and constraints of the MOLA problem. The UML class diagram of its structure is shown in Figure 7.1. The problem is stored as a singleton, which can be accessed by any component of the algorithm at any time to view its objectives and constraints. Currently, the objectives and constraints being optimized are read from the program settings when calling an initialization function. Optionally, this can be automated in the constructor function.

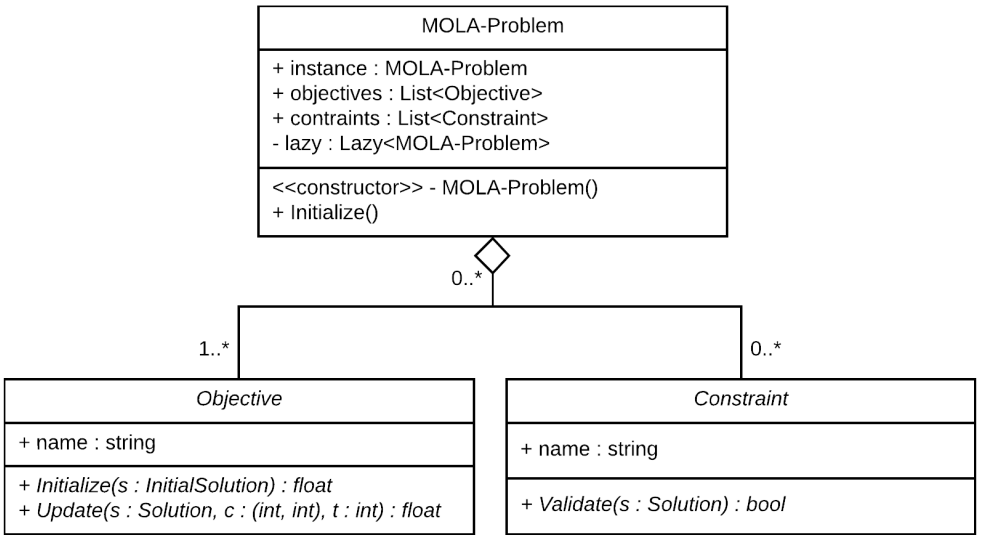


FIGURE 7.1: UML Diagram: Problem Structure

The problem singleton contains both the objectives and constraints. An objective has two functions; an initialization function and an update function. The first calculates the objective function value of a solution from scratch. The second calculates the change in objective function value, given a potential change to a solution. This allows for an incremental objective function handling. A constraint only has a validate function, that given a solution, decides whether it satisfies the constraint. Doing so efficiently, can for example be done using the metadata structures as was discussed in Chapter 6.2.3. The objective and constraint function are coded as abstract classes, completely decoupling the framework from the specific objectives or constraints being used.

7.2.2 Solution Storage

The manner in which solutions to the MOLA problem being optimized are stored, influences the memory requirements of the optimization program. Throughout an optimization run, with either PLS or NSGA-II, there are generally a large number of solutions that are being generated. The number of solutions stored in the archive or population can be high, and on top of these new solutions are being created (possibly in a concurrent manner), asking for even more memory space to be available. Storing all solutions independently and in a naive manner is therefore considered to be unacceptable when developing a scalable optimization program. As far as concerned, no previous MOLA optimization researches have proposed a structure to do so more efficiently. Therefore, a data structure for MOLA solutions has been designed that solely stores changes to the initial solution, minimizing the amount of memory necessary.

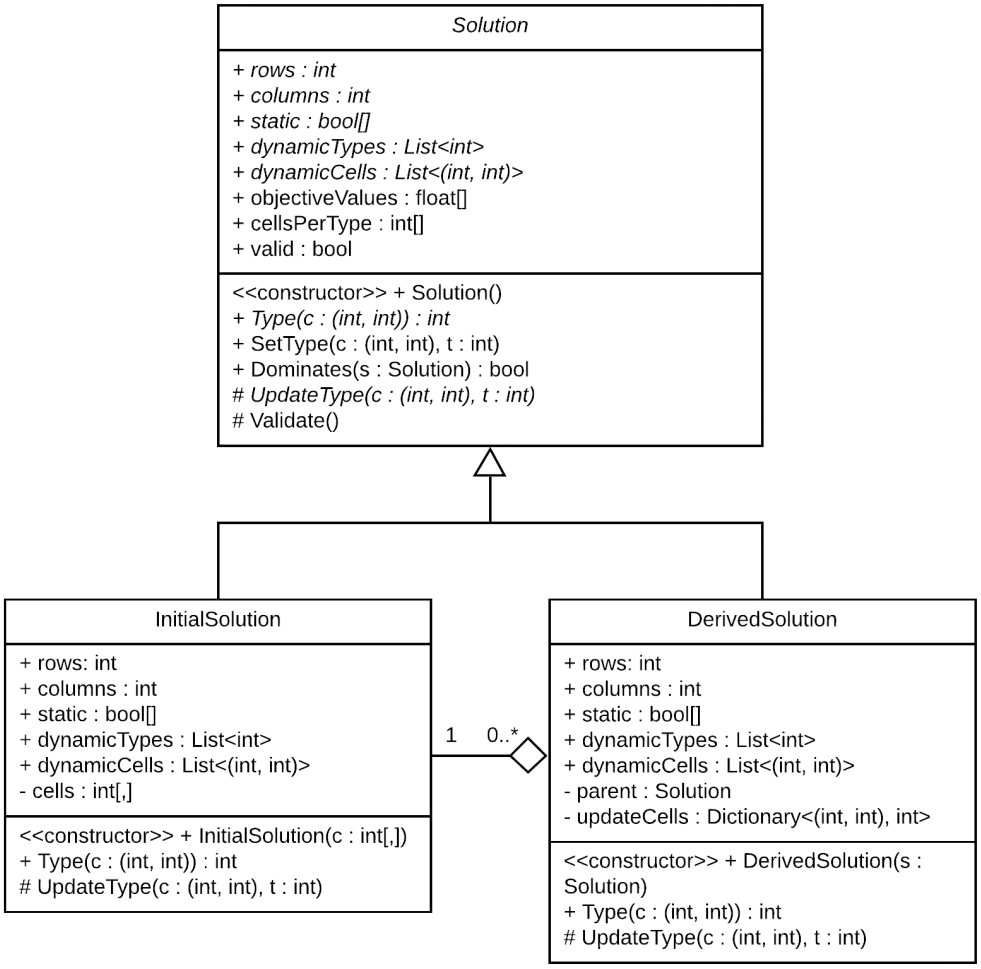


FIGURE 7.2: UML Diagram: Solution Structure

The data structures used for storing a solution are shown in the UML class diagram of Figure 7.2. A solution is defined as an abstract class, with two sub classes; an initial solution and a derived solution. Both provide the same functionalities that are defined in the abstract solution class, meaning that the algorithm does not 'know' with which type of solution it is dealing. However, they both store the cell types in a different manner. An initial solution stores a solution in a naive way; it takes note of the type of every cell of the map. The derived solution however, only stores adjustments made to an initial solution: as so it contains a reference to an initial solution and only takes note of adjusted cell types. Unchanged cell types are now never stored twice, reducing the redundancy. In case the MOLA problem is run with one initial map as starting solution, only one initial solution will have to be stored; all other solutions can be derived solutions based on this initial solution.

The abstract solution class demands both sub classes to have a method to view and update the type of a cell. For the initial solution, these are straightforward. For the derived solution, viewing a cell is done by first checking whether it has an update for this cell; if so, the updated type is returned and otherwise the type in the linked initial solution. When updating a cell, the update is added to its updating list. Furthermore, a solution class takes note of the number of rows, columns and the locations of the dynamic cells. As these properties never change, these are only stored in the initial solution. The derived solutions simply pass on these values when requested. Also, two

metadata structures are stored; the objective value for each objective (MD-I) and the number of cells per land-use type (MD-III). Both of these values do change when an adjustment is made, and so these are updated incrementally when adjustments are made. The derived solution hereby takes note of its own metadata values. As discussed in Chapter 6.2.3, the MD-I and MD-III structures allow for incremental objective updating and constraint checking. Finally, each time a cell type changes, the solution is re-validated to verify whether it still satisfies the problem constraints. The result is stored in the solution itself (the valid property).

7.2.3 Other Features

Several helper classes are set up to provide common functionalities that can be used by any algorithm solving the MOLA problem. These classes are added to Appendix E.1. First of all, the jMetal.NET library (v0.5) is added for hypervolume calculation (Nebro, 2015), which can be used to calculate the hypervolume of an archive or population. Next, a class for IO handling is added, which contains methods for reading and writing solutions from and to .ASC files. Also, this class is able to generate statistics when exporting these solutions, such as the averages and standard deviations of the objective values. Furthermore, a logger class is added that is able to log and print the status of the algorithm in the console. In the future, this could be extended to a form-based interface. Next up, there are two utility classes. One provides general functions such as list shuffling and one provides thread safe random number generators. Finally, a class is added with solution based functionalities. This includes methods that return a random dynamic cell or boundary cell of a solution, that can for example be used by local search operators.

Finally, the resulting framework implementation consisted out of 12 different classes, covering 458 lines of executable code. This excludes sub classes of the abstract objectives and constraint classes. More statistics regarding the framework implementation can be found in Table F.1 of Appendix F. Again, details regarding the exact implementation of each class are out of the scope of this research documentation.

7.3 Algorithms

7.3.1 PLS-MOLA

The PLS-MOLA algorithm is implemented according to the design discussed previously in Chapter 6.3.1. The final UML class diagram is shown in Figure 7.3. First of all, the PLS-MOLA class owns one archive. The archive stores the set of the (currently) best solutions found by the algorithm. It implements one metadata structures (MD-II) to fasten the updating process; a list of each solution sorted by objective value for each objective. This allows for an efficient manner to check whether a new solution is dominated and for crowding distance calculation according to Algorithm 38. Every time a solution is added to the archive, the metadata structure is updated. Finally, the archive also takes note of which solutions are yet unexplored.

In order to explore new solutions, the PLS-MOLA class owns three procedures; a selection procedure, search procedure and repair procedure. These

procedures work according to the algorithms described in Chapter 6.4, Chapter 6.7 and Chapter 6.9 respectively. The selection procedure is responsible for choosing what solution its neighborhood is going to be explored next. The search procedure is responsible for creating new (derived) solutions from this selected solution. Finally, the repair procedure tends to repair a new solution from the search procedure in case it violates any constraints. The PLS-MOLA algorithm is initialized with these three procedures as constructor parameters. This decouples the algorithm from the procedures, allowing the user to for example create and run different versions of the PLS-MOLA algorithm more easily.

Finally, all three procedures possess a set of corresponding operators to realize their task. In Chapter 6.5, Chapter 6.8 and Chapter 6.10, operators were defined for respectively selecting, searching and repairing solutions. These are passed on to the procedures when initialized. The selection and search procedure both require at least one operator to be present; these are necessary for the PLS-MOLA algorithm to run. The repair procedure does not necessarily need to have a repair operator, as this is unnecessary in case the problem has no constraints. All operators are defined as abstract classes, allowing the user to easily add or remove operators from and to the program.

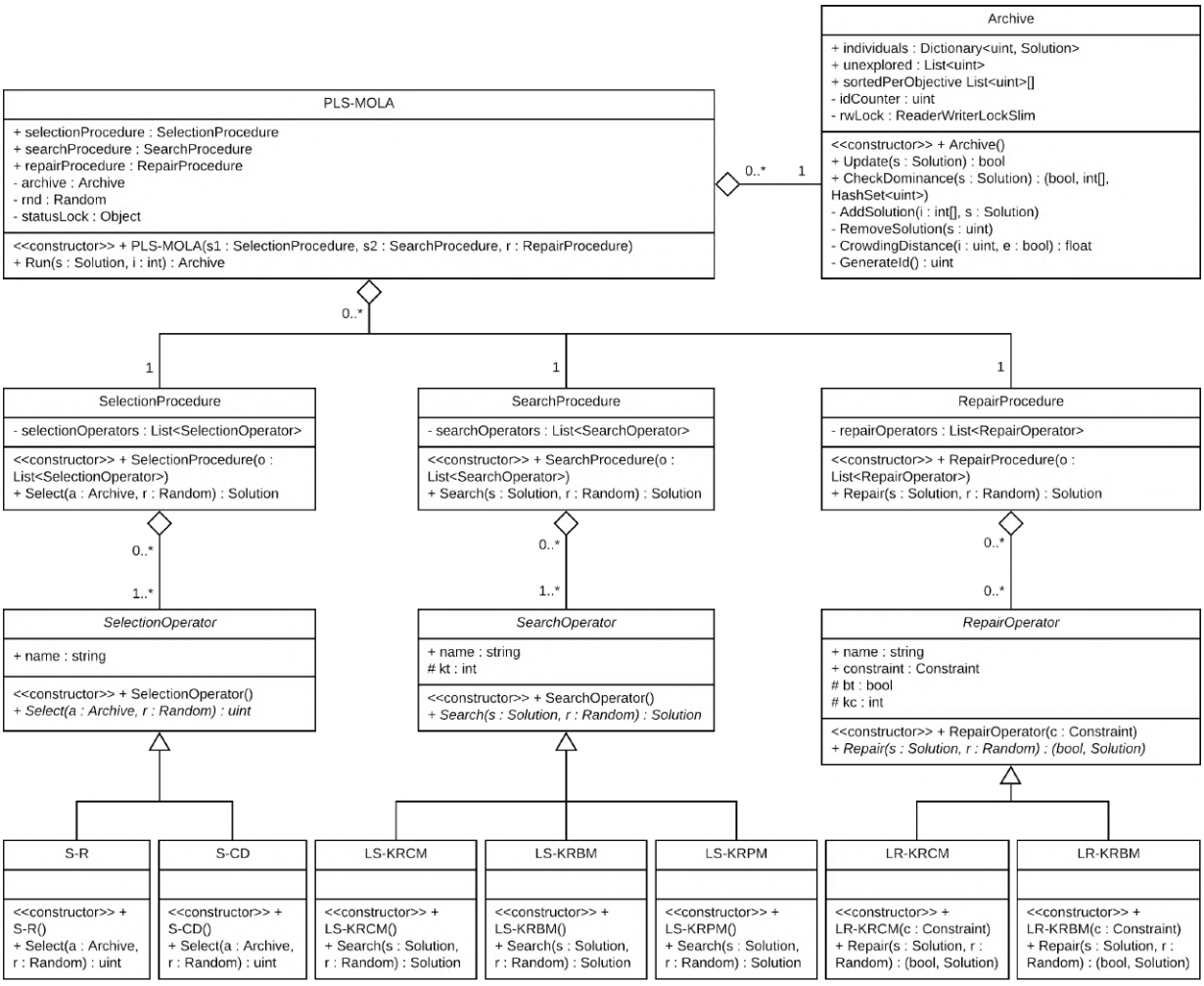


FIGURE 7.3: UML Diagram: The PLS-MOLA Algorithm

Statistics regarding the resulting PLS-MOLA implementation can be found

in Table F.1 of Appendix F. The PLS implementation consists out of a total of 16 classes: all classes of the UML Diagram of Figure 7.3 and one validator class. The validator class validates the problem and PLS settings that have been set by the user before running the algorithm and is shown in Figure E.7. Eventually, the 16 classes cover a total of 309 lines of executable code.

7.3.2 IPLS-MOLA

The local search algorithm PLS-MOLA can be modified to become the global search algorithm IPLS-MOLA. The UML diagram of this algorithm is shown in Figure 7.4. The IPLS algorithm aggregates a PLS-MOLA algorithm, and calls this algorithm in each of its iterations. At the end of an iteration, a solution is perturbed to create a new initial solution for the next iteration. In order to perturb a solution for a new iteration, the IPLS algorithm randomizes a part of the solution. After each iteration, the archive of IPLS-MOLA is updated with the archive found in the latest iteration. Eventually, the final archive containing the best non-dominated solutions found in all iterations is returned. Overall, the implementation follows the general algorithm definition of IPLS as was described in Chapter 5.2.2. No notable adjustments or improvements were made.

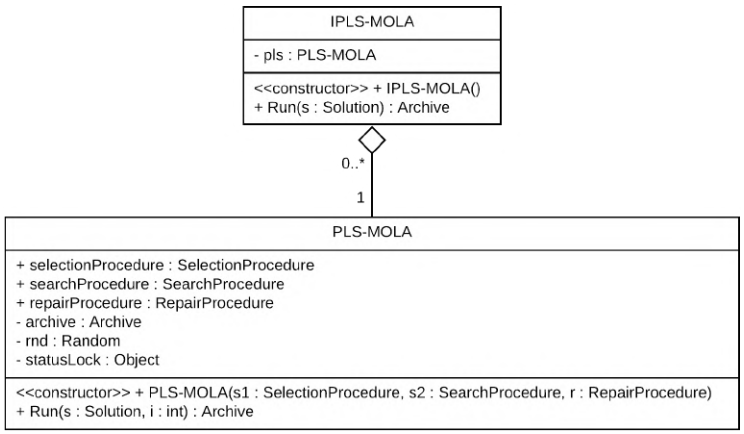


FIGURE 7.4: UML Diagram: The PLS Algorithm

The IPLS-MOLA algorithm only requires one more class on top of the PLS-MOLA classes with 45 lines of executable code. More details on the class statistics can be found in Table F.1 of Appendix F.

7.3.3 NSGA-II-MOLA

The final algorithm implementation to be discussed, is the NSGA-II-MOLA algorithm. The UML class of the implementation is shown in Figure 7.5. In contrast to the archive of PLS, the NSGA-II algorithm owns a population class that aggregates the solutions. This class also contains the methods that are essential for NSGA-II; a fast non-dominated sorting algorithm and a procedure that selects the next population based upon their domination front and crowding distance. The fast non-dominated sorting algorithm is shown in Algorithm 2 of Chapter 4.3.5. In order to perform this sort efficiently, a metadata structure has been added to the population that takes note of which solution(s) dominate or are dominated by what other solution(s). Storing these, makes it unnecessary to check dominance between any two solutions

twice. Similar to the archive class, the population class also contains a list of each solution sorted by objective value for each objective (MD-II). This can (similar to with PLS-MOLA) be used to efficiently calculate the crowding distances. Using the fast non-dominated sorting algorithm, combined with crowding distance comparing for solutions of similar fronts, the population class is able to select the next population. Doing so, is triggered by the NSGA-II-MOLA algorithm, after the desired number of offsprings have been added to the population. The population class will then bring back the number of solutions to the desired population size, using the common NSGA-II selection procedure.

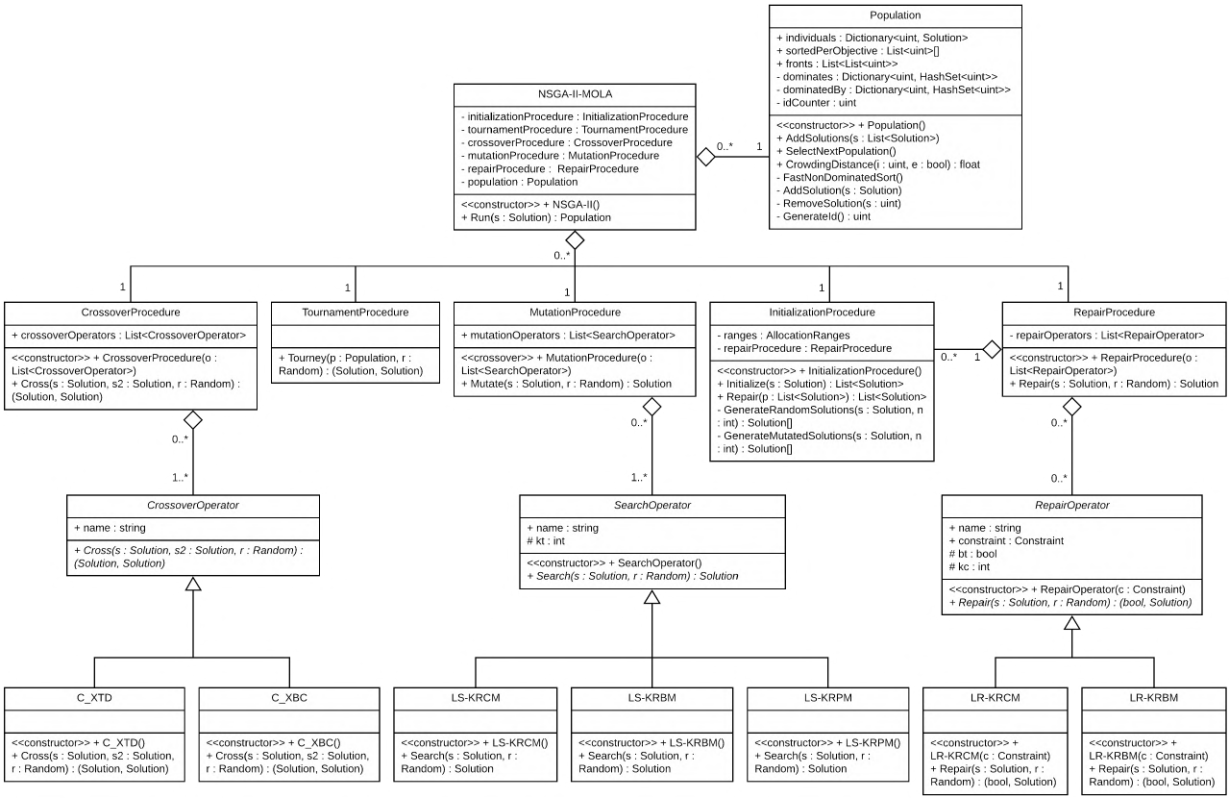


FIGURE 7.5: UML Diagram: The PLS Algorithm

Similar to the PLS-MOLA algorithm, the NSGA-II-MOLA algorithm aggregates several procedure classes to realize the different processes of the algorithm. First of all, the algorithm has an initialization procedure. The initialization procedure initializes the first population, based upon the initial solution that is given to the algorithm. The population is initialized by creating completely random solutions and/or mutating (randomizing) parts of the initial solution. These are the two most common initialization strategies, as was discussed earlier in Chapter 4.3.4. The user can decided upon what part (percentage) of the initial population will be randomized, and what part will be mutations of the current situation. Besides this, the user is able to set the size of these mutations. The size is set as the percentage of dynamic cells that will obtain a random type. The initialization procedure also aggregates a reparation procedure. This procedure is used to eventually repair all newly created solutions and end up with a valid initial population. In this implementation, both initialization strategies already take the allocation ranges constraint (see Chapter 3.2) in account in case it is added to the problem. This

is not necessary, but prevents the created solutions from deviating too much from the allowed ranges, shortening the reparation times.

The initialization procedure is used once to create the initial population. Afterwards, the NSGA-II-MOLA algorithm continuously uses four other procedures to create new generations. First of all, a tournament procedure is used. This procedure performs two tournament selections with k = 2 on the current population to end up with two new parents. A tournament selection is won by the solution in the highest domination front or with the highest crowding distance in case these are equal. After the parents have been selected, the crossover procedure is used to generate offsprings. The crossover procedure uses crossover operators that create two offsprings out of two parents. Two crossover operators are implemented; the C-XTD operator, which is equal to the TDX operator of Chapter 4.3.6, and the C-XBC operators, which is equal to the BCX-III operator of Chapter 4.3.6. Next, both offsprings are mutated using the mutation procedure which applies mutation operators a pre-defined number of times to a solutions. In this case, local search operator that were previously used in PLS-MOLA, are now used as mutation operators. As discussed in Chapter 6.8, these local search operators were originally based upon mutation operators from previous research. Finally, a repair procedure equal to the one used in PLS-MOLA is added to repair invalid offsprings (Chapter 6.9). This procedure also aggregates and applies the same repair operators as were discussed in Chapter 6.10.

Finally, several extensions were added to the framework to provide functionalities specifically needed for NSGA-II-MOLA. These can be viewed in Appendix E.2. The extensions allow the user to export population statistics, log specific NSGA-II details and validate settings of the NSGA-II algorithm. Statistics regarding the final NSGA-II-MOLA related classes can be found in Table F.2. In total, 12 classes were added to the existing PLS and framework classes to realize the NSGA-II-MOLA algorithm. In total, these cover up to 392 lines of executable code.

7.4 Parameters

In this section, a small summary will be given with all parameters that can be set when running the program. The parameters are mostly parameters of earlier discussed algorithms, operators and termination criteria. First, parameters regarding the MOLA problem will be discussed; these always need to be set. Next up, algorithm specific parameters will be discussed; these only need to be set in case the corresponding algorithm is executed.

The first parameters to be discussed are used to set the problem, and can be viewed in table 7.1. The first two parameters decide upon the objectives and constraints. At least one objective is required to run the optimization process. Next, the number of different types in the initial solution, including a list of which of these are static, need to be set. Finally, the problem can be set to either a four- or eight-neighbor problem.

TABLE 7.1: Problem Parameters

Id	Parameter	Type	Size/Range
1	Objectives	{Objective}	$[1, \infty)$
2	Constraints	{Constraint}	$[0, \infty)$
3	Number of Types	int	$[2, \infty)$
4	Static Types	{int}	$[0, \infty)$
5	Number of Neighbors	int	[4, 8]

Next up, are the parameters regarding the PLS-MOLA algorithm, shown in Table 7.2. First of all, the user is able to set the selection, search and repair operators. As discussed earlier, at least one selection and search operator is required. The Search KT parameter sets the type tournament size of the search operators, whereas the Repair KT sets the cell tournament size of the repair operators. These are equal to the k_t and k_c parameters of Chapter 6.8 and 6.10 respectively. The Repair BT parameter is an extra parameter added to the repair operators, that states whether the type resulting in the highest positive objective change should be chosen (BT = true) or a randomly selected type (BT = false). Furthermore, the Allow Repair parameter decides whether or not the PLS-MOLA algorithm should allow the reparation of invalid new solutions, or should simply discard invalid solutions and create new ones until a valid one shows up. The maximum archive size decides upon the maximum size of the archive; solutions with the lowest crowding distance are discarded when the maximum size is exceeded (Chapter 6.11.3). Finally, the exploration, reparation and PLS N and M values of the termination criteria of the PLS-MOLA algorithm (Chapter 6.3.1, Chapter 6.6 and Chapter 6.9) are added.

TABLE 7.2: PLS-MOLA Parameters

Id	Parameter	Type	Size/Range
6	Selection Operators	{SelectionOperator}	$[1, \infty)$
7	Search Operators	{SearchOperator}	$[1, \infty)$
8	Repair Operators	{RepairOperator}	$[0, \infty)$
9	Search KT	int	$[1, \infty)$
10	Repair KC	int	$[1, \infty)$
11	Repair BT	bool	[True, False]
12	Allow Repair	bool	[True, False]
13	NDS	bool	[True, False]
14	Max. Archive Size	int	$[1, \infty)$
15	Exploration N	int	$[0, \infty)$
16	Exploration M	int	$[0, \infty)$
17	Reparation N	int	$[0, \infty)$
18	Reparation M	int	$[0, \infty)$
19	PLS N	int	$[0, \infty)$
20	PLS M	int	$[0, \infty)$

The parameters regarding the IPLS-MOLA algorithm are shown in table 7.3. Since the IPLS algorithm makes use of the PLS algorithm, the previously discussed PLS settings need to be set as well; the two IPLS parameters simply expand this parameter set. The first extra parameter states the perturbation size (Chapter 6.14), whereas the second parameter states the number of iterations of the IPLS-MOLA algorithm. The perturbation size is the percentage

of the solution (dynamic cells) that will be randomized in the perturbation process, and should be in between 0 and 100.

TABLE 7.3: IPLS-MOLA Parameters

Id	Parameter	Type	Size/Range
21	Perturbation Size	int	$[0, \infty)$
22	Iterations	int	$[1, \infty)$

The fourth and final set of parameters that will be discussed, is related to the NSGA-II-MOLA algorithm. At first, the crossover, mutation (search) and repair operators need to be set. These are followed up by the population size and two parameters that define the formation of the initial population. The first one is the Init. Random Sol. parameter, which decides what part (percentage) of the initial population should consist of completely randomized solutions. The remaining part will consists of mutations of the initial solution. The second parameter, is the Init. Mut. Size parameter, which decides the mutation size of these mutated solutions. The size is defined as what part (percentage) of the dynamic cells of the initial solution should be randomized. Note, that this is not equal to the part of cells that is eventually updated; a randomization might lead to the same type as before. Both parameters should be in between 0 and 100. Next up, the mutation parameters related to the mutations that occur during the optimization process need to be set. The mutation N is the mutation size, and states the number of times a local search operator is applied to a solution being mutated. The Mutation KT is equal tot the Search KT parameter of parameter 9. The repair KC and repair BT are also equal to the repair KC and repair BT of parameter 10 and 11. Finally, the termination criteria are set. The first two set the N and M for the C-XBC crossover operator. The third and fourth set the N and M for the reparation procedure. Finally, the number of generations that the NSGA-II-MOLA algorithm will run are set using the generation parameter.

TABLE 7.4: NSGA-II-MOLA Parameters

Id	Parameter	Type	Size/Range
23	Crossover Operators	{CrossoverOperator}	$[1, \infty)$
24	Mutation Operators	{SearchOperator}	$[0, \infty)$
25	Repair Operators	{RepairOperator}	$[0, \infty)$
26	Population Size	int	$[1, \infty)$
27	Init. Random Sol. (%)	int	$[0, \infty)$
28	Init. Mut. Size (%)	int	$[0, \infty)$
29	Mutation N	int	$[1, \infty)$
30	Mutation KT	int	$[1, \infty)$
31	Repair KC	int	$[1, \infty)$
32	Repair BT	bool	[True, False]
33	XBC N	int	$[0, \infty)$
34	XBC M	int	$[0, \infty)$
35	Reparation N	int	$[1, \infty)$
36	Reparation M	int	$[1, \infty)$
37	Generations	int	$[0, \infty)$

7.5 Concurrency

Finally, in order to speed up the optimization processes of the algorithms, independent parts of the procedures can be run in parallel. However, it should hereby be assured that the outcome of one thread does not influence the outcome of another thread. In this section, we will shortly discuss which parts of the algorithms can be executed in parallel and in which parts concurrency should be avoided.

First of all, let us discuss parallelism regarding the PLS-MOLA algorithm. When looking at the PLS-MOLA algorithm in Algorithm 15, we see two loops that could be executed in parallel. The first one is related to the exploration of a single solution. Doing so in parallel, would result in the exploration of multiple solutions of the archive simultaneously. However, the outcome of the exploration of one solution could affect the need to explore another solutions in the current archive. New and better solutions can be found, that would provided better 'starting solutions' for the next exploration. As the PLS algorithm only runs for a pre-set number of iterations, it is desired to only 'spend' iterations on explorations of the best solutions possible. Therefore, it is undesired to execute this loop in parallel. The second loop considers the exploration of one neighborhood. In this neighborhood exploration, a pre-set number of neighborhood solutions will be search for and evaluated, independently of each other. By doing so in parallel, the exploration of one single neighborhood can be sped up without affecting the outcome.

Secondly, let's discuss parallelism regarding the IPLS-MOLA algorithm. At first, it might look as if multiple iterations (calling the PLS-MOLA algorithm) can be done in parallel. However, similar as to the exploration of multiple solutions in the archive, these iterations depend on each other. After each iteration, the quality of the archive of the IPLS-MOLA algorithm rises, providing a better start for the next iteration. However, as the PLS-MOLA algorithm already exploits multiple threads in the exploration of one solution, IPLS-MOLA automatically makes use of concurrency as well.

Thirdly, the NSGA-II-MOLA algorithm can exploit parallelism in two processes. First of all, both the generation and reparation of the initial population can be done in parallel. As these solutions need to be created and repaired independently of each other, concurrency can speed up both processes. Secondly, the expansion of the population with new offsprings can be done in parallel. For each generation, the number of solutions in the population needs to be doubled. Creating and repairing new offsprings can be done independently, and so this can efficiently be done in parallel as well.

Finally, the framework exploits parallelism when exporting the final solutions to .ASC files. As shown in Table 7.5, the program can both be set to run in serial or parallel. As parallelism does not affect the solution quality of any algorithm, but only decreases the time necessary to complete certain operations, all algorithms will be experimented with while allowing multiple threads. Meaning, that all results of Chapter 9, were generating using the concurrency optimizations discussed above.

TABLE 7.5: Other Parameters

Id	Parameter	Type	Size/Range
38	Asynchronous	bool	[True, False]

Chapter 8

Experimentation

8.1 Experimental Setup

In order to answer the research questions defined at the start, several test cases has been set up. Each test case includes multiple experiments designed to answer (part of) a research question and/or give more insight in the overall efficiency of PLS for MOLA. Eventually, this resulted in a total of 8 test cases, described in Chapter 8.3. First, multiple tests cases are set up to optimize the PLS-MOLA algorithm, as was stated by RQ1. Next, the influence of the number of objectives on the performance of PLS-MOLA (RQ2) and optimization of IPLS-MOLA (RQ3) will be examined. Finally, the performance of (I)PLS-MOLA will be compared to NSGA-II in the eighth test cases. However, before the test cases are described, a present-day MOLA case will be set up to optimize in the experiments. As was discussed in Chapter 1, this case will be set up in cooperation with Dr. F. van der Hilst of the Energy and Resources group of Utrecht University. Van Der Hilst worked on multiple (international) research projects concerning land use changes due to biomass production, and their effects on the environment. One of the cases Van Der Hilst has been working on, involves the direct and indirect land-use changes in Brazil in relation to several sustainability concerns. As the Brazil case is a topical land-use optimization problem, it will be used as the case study in the experiments. The Brazil case is discussed in more detail in Chapter 8.2.

8.2 Problem Case: Brazil

8.2.1 Problem Context

The use of biomass for bio-energy is considered an important option to reduce the dependency on fossil resources and mitigate climate change. Among others, it is able to reduce greenhouse gas emissions and contribute to economic development (Hilst, Verstegen, Woltjer, et al., 2018). Over the last few years, many governments have set biofuel targets. The increase in demand of biomass for energy and materials has led to several sustainability issues. Most of these are caused by land-use changes, which can either be direct or indirect. Direct land-use changes occur for instance when agricultural land use is converted to biofuel feedstock to meet the rising biofuel demands. Indirect land-use changes occur for instance when the direct conversions alter the price of agricultural products, resulting in an increase in the amount of agricultural land elsewhere (Wicke, Verweij, Meijl, et al., 2012). These land-use changes are likely to have environmental and socioeconomic effects, such as an impact on carbon stocks, biodiversity, water availability and soil quality as well as on rural development and food security. Examples of these, are deforestation (of for instance the Amazon) and a decrease in the quality of

the soil or biodiversity. (Hilst, Verstegen, Woltjer, et al., 2018). Therefore, the expansion of biomass production requires monitoring and good governance to reduce these negative effects. Part of this good governance, includes the development of effective policy strategies. Being able to explore how these expansions can be done in a sustainable manner, could help in this debate.

An interesting area to examine biofuel-induced land-use changes, is Brazil. First of all, Brazil plays a crucial role in the production and global supply of ethanol. Considering ethanol, the annual production increased from 11.5 x 10^9L in 1990/91 to 30.2 x 109L in 2015/16 (Hilst, Verstegen, Woltjer, et al., 2018). Projections of future ethanol production expect this amount to increase even further to meet the upcoming global demands. Brazil is also an important producer and exporter of agricultural products and wood (fibre). Due to the abundance of natural resources and favourable climate conditions, the agricultural sector has the potential to expand significantly in the upcoming years. These strong developments in the agricultural and biofuel sector in Brazil have several positive effects, such as economic growth and rural development (Walter, Galdos, Scarpare, et al., 2014). However, it also results in land-use change related greenhouse gas emissions and loss of ecosystems such as the Amazon, Cerrado and Atlantic forest (Martinelli, Naylor, Vitousek, et al., 2010).

The situation in Brazil can be translated into a multi-objective land-use allocation problem. The essence of this combinatorial problem will be to find out how the land-use of Brazil can be allocated in a sustainable manner while satisfying the global food, feed and biofuel demands. Although many objectives can be thought of to steer towards the optimal situation, this research will be limited to three objectives; compactness, potential yield and carbon stock maximization. Compactness promotes the creation of less 'scattered' solutions, potential yield is a proxy for profit/economic viability of the production related land-use types and carbon stock states the amount of carbon (related to greenhouse has emissions) being stored. In Chapter 8.2.3, we will elaborate more on these objectives and why these were considered as the most interesting. Finally, the solutions will have to meet the expected production demands in food, feed, fibre and biofuel of 2030, which will be the constraint of the problem. In the next chapter, we will describe in more detail how the situation of Brazil can be translated into an optimization problem. In Chapter 8.2.3 and Chapter 8.2.4, the three objectives and constraint will be discussed. Eventually, the necessary resources and data will be handled in Chapter 8.2.6.

8.2.2 Problem Definition

In order to optimize land-use allocation in Brazil, the situation of 2012 has been translated into a grid with the corresponding land-use types. The resulting land-use grid is shown in Figure 8.1. The size of one grid-cell is 5 x 5 km. Eleven different land-use types have been distinguished and are summarized in Table 8.2. Two of these land-use types are static, whereas the other nine are dynamic. The right columns show the area being covered and the number of cells per land-use type, including their percentage of the total number of cells. In total, the map of Brazil exists out of 342815 cells (8570375 km^2).

FIGURE 8.1: Initial Map of Brazil

TABLE 8.1: Land-Use Types Brazil

Id	Type	Static	Km^2	Cells	
1	Urban	Yes	5900	236	(0.1%)
2	Water	Yes	161925	6477	(1.9%)
3	Natural Forest	No	4529000	181160	(52.8%)
4	Rangeland	No	239525	19581	(5.7%)
5	Planted Forest	No	142800	5712	(1.7%)
6	Crops	No	405750	16230	(4.7%)
7	Grass & Shrubs	No	1524000	60960	(17.8%)
8	Sugar Cane	No	99550	3982	(1.2%)
9	Planted Pasture	No	1017375	40695	(11.9%)
10	Bare Soil	No	6500	260	(0.1%)
11	Abandoned	No	188050	7522	(2.2%)

However, there are several areas in the map of Figure 8.2 of which it is assumed that the land-use will not change (for various reasons). These areas can be found among all different land-use types. The initial map can therefore be simplified, by removing these areas from the map beforehand. In addition, we can also remove the static types Urban (1) and Water (2), as these are assumed not to change either. The resulting initial map, that will serve as the starting point of the optimization process, is shown in Figure 8.2. By

removing 236 Urban cells, 6477 Water cells and 57630 so-called no-go cells, a total of 278472 dynamic cells remain.

FIGURE 8.2: Initial Map of Brazil with No-Go Areas

TABLE 8.2: Land-Use Types Brazil with No-Go Areas

Id	Type	Static	Km^2	Cells
3	Natural Forest	No	3228875	129155 (46.4%)
4	Rangeland	No	477050	19082 (6.9%)
5	Planted Forest	No	140500	5620 (2.0%)
6	Crops	No	404825	16193 (5.8%)
7	Grass & Shrubs	No	1408800	56352 (20.2%)
8	Sugar Cane	No	99550	3982 (1.4%)
9	Planted Pasture	No	1012125	40485 (14.5%)
10	Bare Soil	No	5275	211 (0.1%)
11	Abandoned	No	184800	7392 (2.7%)

The Brazil case can be translated into a MOLA problem by rewriting the MOLA definition given in Chapter 3.2. The choice has been made to maximize the objectives (similar to minimizing their negative version). Furthermore, the grid of the Brazil map has 885 rows and 854 columns, and type 3 to 11 are dynamic. Hereby, the static type 12 is added to represent all static types (1 and 2) together with all cells that do not below to Brazil (white space). The

MOLA problem can now be formulated as follows;

Maximize:

$$\sum_{k=3}^{11}\sum_{i=1}^{885}\sum_{j=1}^{854} B_{ijk}x_{ijk} \tag{8.1}$$

Subject to:

$$\sum_{k=3}^{12} x_{ijk} = 1 \quad \forall i = 1,...,885 \quad j = 1,...,854 \tag{8.2}$$

$$L_k \leq S_k \leq U_k \tag{8.3}$$

$$\sum_{i=1}^{885}\sum_{j=1}^{854} x_{ijk} = S_k \quad \forall k = 3,...,11 \tag{8.4}$$

where:

$$x_{ijk} \in \{0,1\} \quad \forall i = 1,...,885 \quad j = 1,...,854 \quad k = 3,...,12 \tag{8.5}$$

Equation (8.1) can occur multiple times; once for each objective. Each objective hereby sets different values for B_{ijk}, as will be discussed in Chapter8.2.3. Note, that an objective only takes land-use type 3 to 11 in account. Constraint (8.2) states that each cell either has to be of type 3 to 11 or type 12 (static or outside Brazil). Finally, constraint (8.3) and (8.4) state the desired number of cells per land-use type in 2030. These should be in between a lower bound L_k and upper bound U_k for each type $k = 3,...,11$. This will be further discussed in Chapter 8.2.4. Optionally, the problem could be formulated using a vector-based representation as was discussed in Chapter 3.5 (for example to decrease redundancy). However, since the PLS-MOLA algorithm is designed for the (more convenient) grid-based MOLA representation (Chapter 6.2.2), the Brazil case is formulated in this manner as well.

8.2.3 Objectives

In the research of Verstegen, Hilst, Woltjer, et al., 2016 and Hilst, Verstegen, Woltjer, et al., 2018, future land-use changes in Brazil were estimated using so-called suitability factors. These factors can be translated into objectives, as they tend to minimize or maximize certain cell properties. The suitability factors included agro-ecological suitability, the distance to road infrastructures, the travel time to processing hubs, the availability of the same land-use in the neighbourhood and the conversion elasticity. The availability of the same land-use in the neighborhood is comparable with the compactness objective of Chapter 3.3.3, whereas the conversion elasticity is comparable with cost minimization of Chapter 3.3.1. Note, that these suitability factors were used to estimate future changes, whereas the objectives will be used to 'optimize' these changes.

For this case, three of the suitability factors have been transformed into objectives. Handling more than three objectives is out of the scope of this research, and so three factors were selected that are considered to be the most

interesting when optimizing land-use change in Brazil. The suitability factors that have been selected, are the availability of the same land-use in the neighborhood (compactness), the potential yield and the loss of carbon stock. Compactness was selected in order to prevent solutions from creating a high amount of small clusters. Potential yield was chosen as it is a proxy for profit/economic viability, which is assumed to be an important motivation of land-use change. Finally carbon stock was chosen, to be able to examine how these changes can be done in a sustainable manner. In the following sub chapters, the three selected suitability factors and resulting MOLA objectives will be discussed in more detail. Note, that it should be possible to update an objective value incrementally, as was one of the requirements when using PLS-MOLA. All three objectives that were set up, satisfy this requirement.

8.2.3.1 Compactness

The first objective is formed out of the suitability factor concerning the availability of the same land-use in the neighbourhood. It is preferred to have similar land-use types nearby, in order to prevent similar land-uses from being scattered (to a great extent). The neighborhood will be defined as the 4 or 8 cells adjacent to a cell. In the experimentation with MOLA, we will work according to the 4-cell neighborhood (as this is the most simple form), meaning that parameter 5 (number of neighbors) of Chapter 7.4 will be set to 4. The objective used to stimulate nearby land will be based on the compactness objective as defined in Algorithm (3.13) and (3.14) of Chapter 3.3.3. The compactness objective for the Brazil case can then be defined as follows:

Maximize:

$$\sum_{k=3}^{11}\sum_{i=1}^{885}\sum_{j=1}^{854}\left(x_{ijk}\sum_{m=i-1}^{i+1}\sum_{n=j-1}^{j+1}\frac{Neig_{mn}}{4}\right) \tag{8.6}$$

where

$$Neig_{mn}=\begin{cases}1, & \text{if } x_{ijk}=x_{mnk}\\ 0, & \text{otherwise}\end{cases} \tag{8.7}$$

8.2.3.2 Potential Yield

The second objective concerns the potential yield, which is a proxy for the economic viability. For this objective, four yield maps have been created for five different land-use types; one for Planted Forest (5), one for Crops (6), one for Sugar Cane (8), and one for both Rangeland (4) and Planted Pasture (9). More details on the resources and references upon which these maps have been based, can be found in Verstegen, Hilst, Woltjer, et al., 2016 and Hilst, Verstegen, Woltjer, et al., 2018. The four yield maps, give a score of 0 to 10000 to each cell to indicate the potential yield in case this type is allocated to that cell. Hereby, 0 is the worst and 10000 is the highest potential yield value attainable for one cell. The yield map for the Crops land-use type is depicted in Figure 8.3 for illustration.

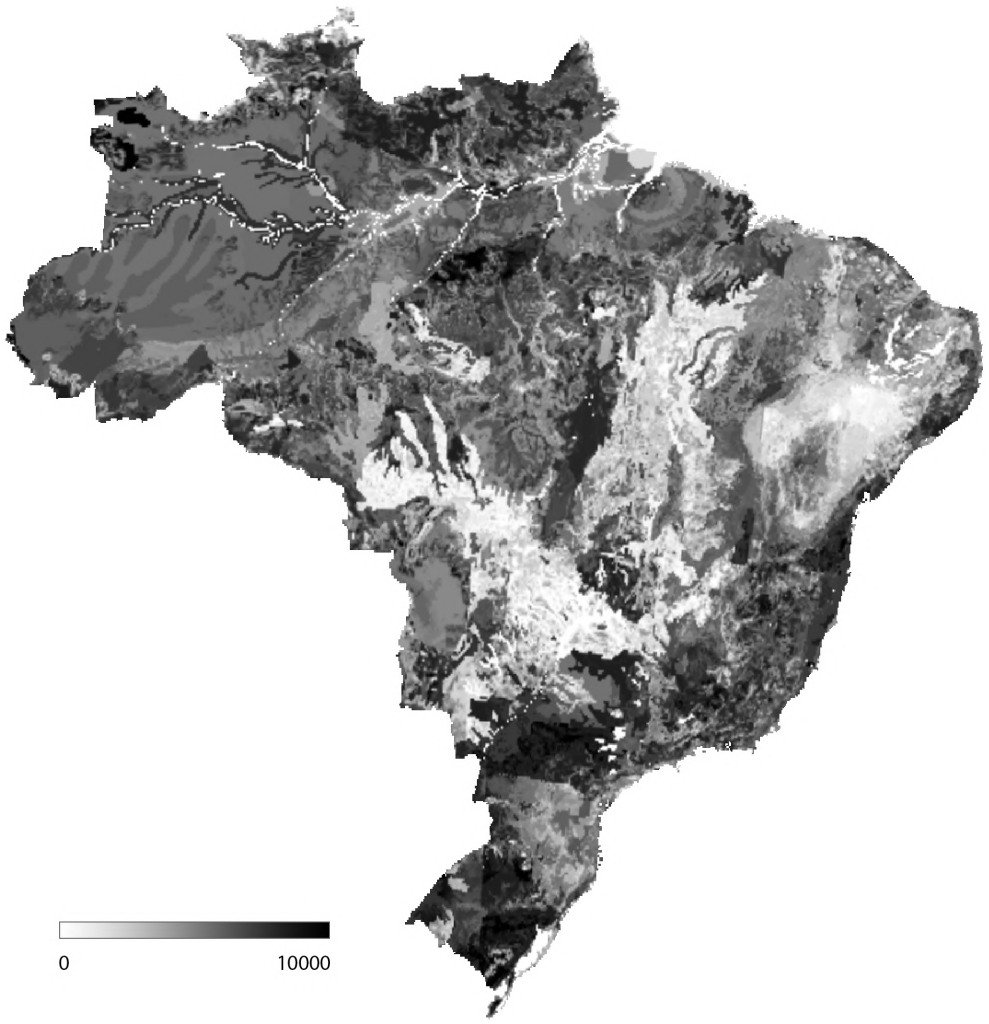


FIGURE 8.3: Potential Yield Map of Crops in Brazil

The total potential yield of the five previously named types should be maximized. The remaining land-use types do not contribute to the potential yield, and have a potential yield of 0 assigned to each cell. Land-use type 12 is static, and will be neglected in the objective. The objective now becomes a maximization of the potential yield for all dynamic land-use types, formulated as follows;

Maximize:

$$\sum_{k=3}^{11} \sum_{i=1}^{885} \sum_{j=1}^{854} Y_{ijk} x_{ijk} \tag{8.8}$$

Where Y_{ijk} is equal to the potential yield (from 0 to 10000) when type $k = 3, ..., 11$ is the type assigned to cell (i, j). For type $k \in \{4, 5, 6, 8, 9\}$, this value can be found in the corresponding cell of its yield map. For all other k, the Y_ijk value is equal to 0 for each cell in the grid. As discussed, an objective optimized by PLS-MOLA is required to be incrementally updatable. Note, that with potential yield maximization this is indeed the case; when the land-use type of a cell is changed, a comparison between the yield of the old and new type reveals the resulting change in objective value.

8.2.3.3 Carbon Stock

The third objective concerns the loss in soil organic carbon and ground biomass, which can result in the emission of carbon dioxide (CO_2) gasses. Minimizing these gas emissions is equal to maximizing the carbon stock. Allocation of a new land-use type changes the amount of carbon stock compared to the current situation. Therefore, carbon stock maps have been created that show the change in carbon stock compared to the current situation in case a certain land-use type is allocated to a cell. The changes in the amount of carbon stock per cell and per type, are given in tonne carbon per hectare. The land-use types Urban (1), Water (2) and Bare Soil (10) have a carbon stock of 0, meaning that an allocation of any of these types results in the negative amount of carbon stock currently present. As Urban and Water are already added to land-use type 12, these will not be taken in account. The initial amount of carbon stock is shown in Figure 8.4.

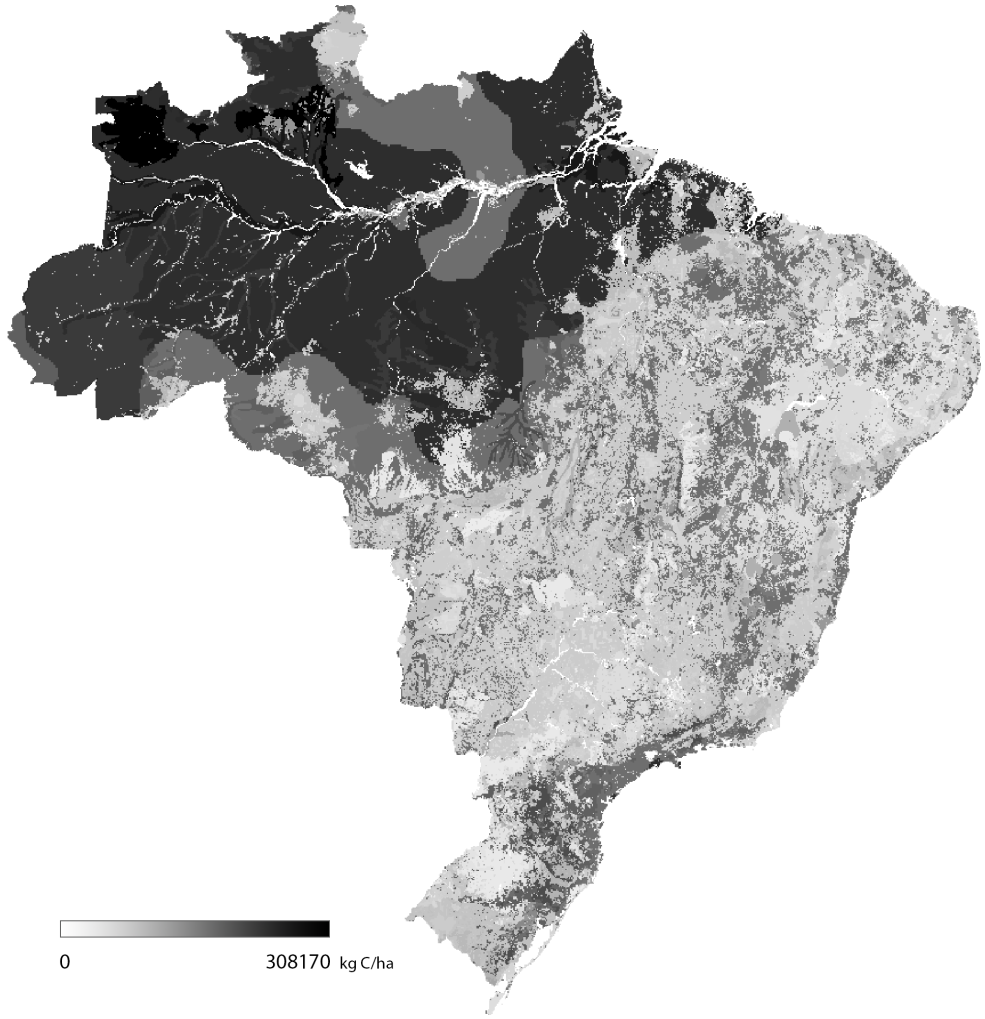


FIGURE 8.4: Initial Carbon Stock Map of Brazil

Preferably, the amount of carbon stock should be as high as possible. As so, the objective will be to maximize the summation of all changes in carbon stock caused by the allocated land-use types. A positive change hereby notes an increase in carbon stock, whereas a negative change states a loss. Again, since land-use type 12 is static it will be ignored in the objective. The carbon stock maximization objective can now be formulated as follows:

Maximize:

$$\sum_{k=3}^{11}\sum_{i=1}^{885}\sum_{j=1}^{854} C_{ijk}x_{ijk} \tag{8.9}$$

Where C_{ijk} is equal to the change in carbon stock compared to the initial solution when type $k = 3, ..., 11$ gets assigned to cell (i, j). These values can be found in the corresponding cell of the carbon map of the type that is being allocated. Again, the objective can be updated incrementally when a cell's type changes, by comparing the change in carbon stock when allocating the old type with the change in carbon stock when allocating the new type.

8.2.4 Constraints

The MOLA case has to obey two constraints, as was shown earlier in Equation (8.2), (8.3) and (8.4). The first constraint is that a land-use type $k = 3, ...12$ has to be allocated to each cell in the grid. The second constraint considers the number of cells that should be allocated to each type to satisfy the predictions of 2030. This will be handled in Chapter 8.2.4.1.

8.2.4.1 Allocation Ranges

In the work of Verstegen, Hilst, Woltjer, et al., 2016 and Hilst, Verstegen, Woltjer, et al., 2018, predictions have been made for the demand in bio ethanol and other agricultural commodities in 2030. These predictions were made using the so-called MAGNET model and translated to the area per (active) land-use type necessary to meet those projected demands. For more details on these models and projections, view Verstegen, Hilst, Woltjer, et al., 2016. The results generated by MAGNET can be used as a constraint for the problem, by requiring each solution to have exact or comparable sizes per land-use type. As so, an optimization can take place of solutions that satisfy the estimated demands of 2030. The resulting areas are shown in Table 8.3. In this case, the areas of the no-go zones have not been deducted from the estimated total area sizes in 2030. Although this would not result in (relatively) large area changes, the actual areas sizes should be slightly smaller.

TABLE 8.3: Area in 2012 and Expected Area in 2030

Type	**2012 (km^2)**	**2030 (km^2)**	**Change**
Rangeland	489451	373760	-23.6%
Planted Forest	142783	168887	+18.3%
Crops	405741	724450	+78.5%
Sugar Cane	99586	136281	+36.8%
Planted Pasture	1017641	1089294	+7.0%

As can be seen in Table 8.3, only five land-use types are relevant for satisfying the estimated global demands in 2030. All other types are allowed to be of any size. Four out of five land-use types require an increase in size, up to 78.5% for crops. On the other hand, rangeland is asked to decrease in size by 23.6%. The size areas can be translated to the corresponding number of cells in the MOLA grid, making the constraint more easy to validate. It is

not required to state an exact size for each land-use type. As was discussed earlier in Chapter 3.2, ranges can be given as well. Ranges are able to make the constraint less 'strict', which could result in less violations and make the algorithm convergence faster. The range size has not been added as a single parameter, as it can be different for each land-use type. Therefore, they are provided using an input file to the implementation of this research. In Table 8.4, an example of the resulting allowed allocation ranges is given, in case the range size of each land-use would be 10 cells. As so, the lower bound is set to the number of cells necessary to realize the estimated area size in 2030 of Table 8.3 minus 5 cells, and the upper bound is set to this number plus 5 cells.

TABLE 8.4: Example Allocation Ranges in Number of Cells when all Range Sizes set to 10

Type	Lower Bound	Upper Bound	Range Size
Rangeland	14567	14577	10
Planted Forest	6642	6652	10
Crops	28908	28918	10
Sugar Cane	5444	5454	10
Planted Pasture	43331	43341	10

The sizes of these ranges influence the performance of the algorithm, as they could increase or decrease the required number of reparations during the optimization process. In Chapter 8.3.2, a comparison will be made between different range sizes (equal for each land-use type) and the time necessary to complete a certain number of iterations. Eventually, it is up to the user to decide what range is acceptable. The range size could also be entered as a proportional value (for example a percentage), in case the user considers this to be more convenient.

8.2.5 Subcases

8.2.5.1 Centre West

The Brazil case is a relatively large case, considering it has over 278472 dynamic cells that take part in the optimization process. In order to also gain insight in the optimization behavior of the algorithms when applied to smaller cases, two subcases have been set up. The first subcase is one out of six macroregions of Brazil, as described in the research of Verstegen, Hilst, Woltjer, et al., 2016. The macroregion, called Centre West, takes an important role in the current and future potential yield of Brazil and is considered to be a relatively good representation of Brazil as a whole. As so, it makes an interesting and suitable subcase for this research. The Centre West region is located in the center of Brazil and shown in Figure 8.5. In contrast to the initial map of Brazil, the Centre West map contains only 63827 dynamic cells (22.9% of Brazil).

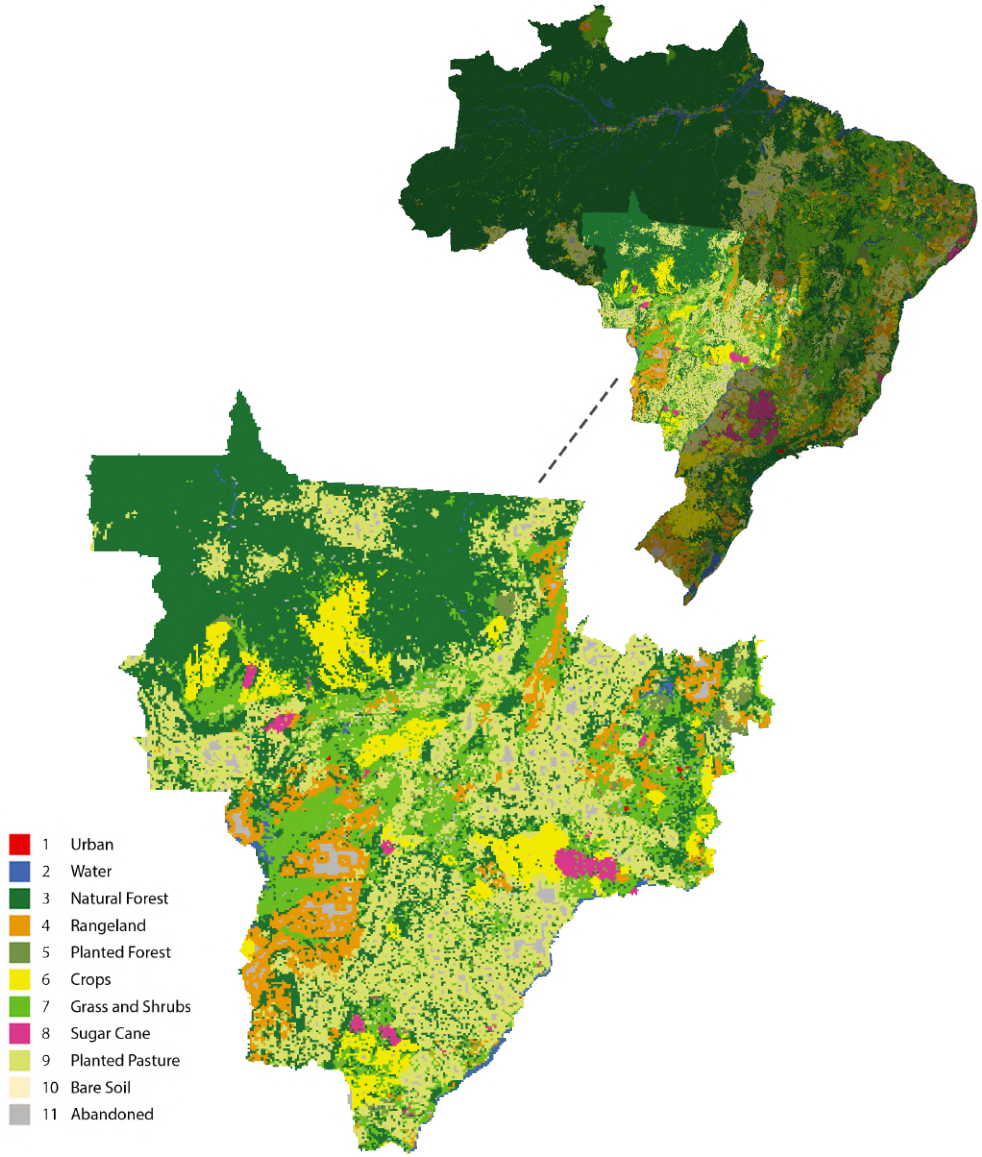


FIGURE 8.5: Centre West of Brazil

The objectives considering the subcase remain equal to the objectives of the complete Brazil case. Instead of the complete potential yield and carbon stock maps, only the values regarding the subcase area are needed. The allocation ranges do change, as these differ per region. The demands in land-use area for Centre West according to Verstegen, Hilst, Woltjer, et al., 2016 and Hilst, Verstegen, Woltjer, et al., 2018, are shown in Table 8.5. Again, the ranges are shown with a size of 10.

TABLE 8.5: Allocation Ranges with Range Size 10 for Microregion 4

Type	Lower Bound	Upper Bound	Range Size
Rangeland	847	857	10
Planted Forest	689	699	10
Crops	12552	12562	10
Sugar Cane	2021	2031	10
Planted Pasture	19449	19459	10

8.2.5.2 Sul Goiano

Finally, a second even smaller subcase has been set up. This subcase considers the mesoregion Sul Goiano and is shown in Figure 8.6. Again, the mesoregion is considered to be a relatively good representation of Brazil as a whole in terms of land-use division. The mesoregion contains 5229 dynamic cells, which is only 1.9% of Brazil.

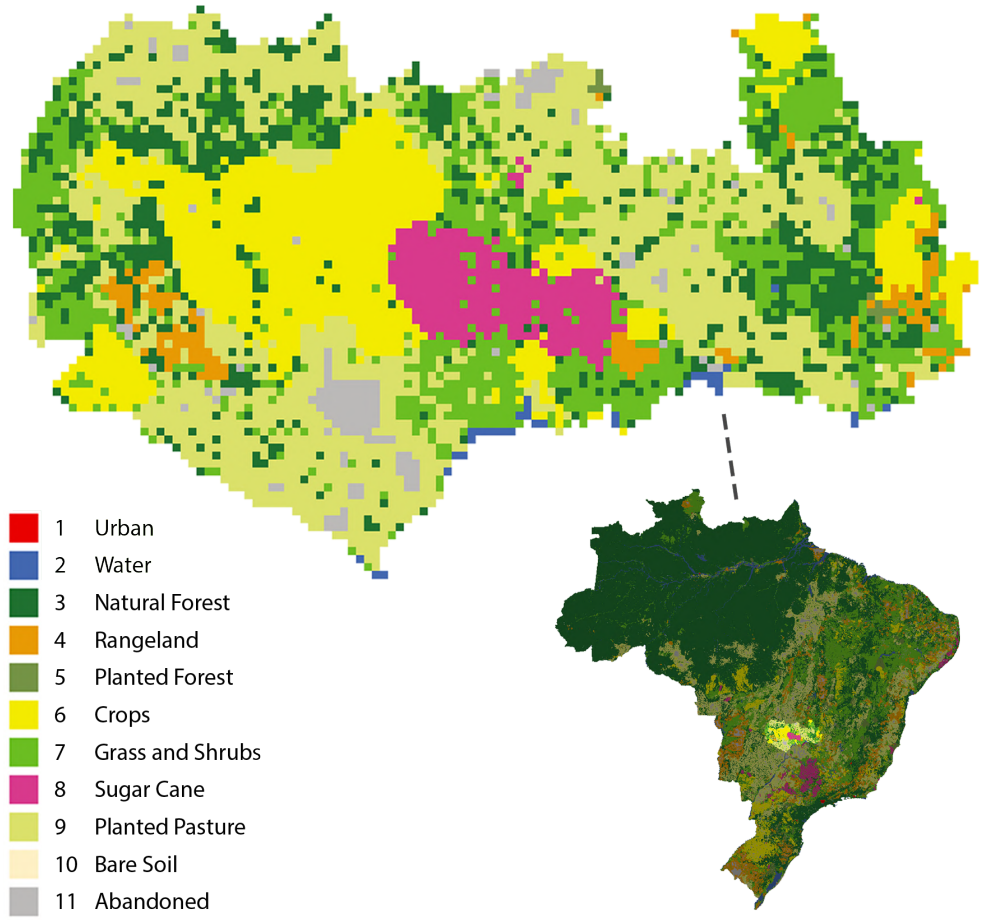


FIGURE 8.6: Mesoregion Sul Goiano of Brazil

Similar to Centre West, Sul Goiano has its own predictions of the number of cells per land-use type in 2030. These are visualised in Table 8.6. The number of expected cells in Rangeland was 0, and so it will range from 0 to 5 as the number of cells can never be negative.

TABLE 8.6: Allocation Ranges with Range Size 10 for Mesoregion Sul Goiano

Type	Lower Bound	Upper Bound	Range Size
Rangeland	0	5	10
Planted Forest	5	15	10
Crops	1167	1177	10
Sugar Cane	784	794	10
Planted Pasture	1946	1956	10

8.2.6 Resources

In order to experiment using the Brazil case and its subcases, several datasets are necessary. First of all, an initial map of the land-uses in Brazil of 2012. This map is necessary as the starting solution for the algorithm. Next, yield maps for crops, sugar cane, planted forest and both rangeland and planted pasture are necessary, in case the potential yield is being maximized. Finally, carbon stock maps for all land-use types are necessary when carbon stock is being maximized. All of these datasets were obtained from the Energy and Resources group of Utrecht University. Obtaining and usage of these datasets should be in consultation with this institute.

8.3 Test Cases

8.3.1 Test Setup

In order to answer the research questions stated in Chapter 1.3, several experiments will be executed. First of all, the experiments compare the performance of different operators, strategies and criteria that can be used in PLS-MOLA and IPLS-MOLA. In order to do so, the archive hypervolume obtained after running the algorithm with different settings/components for an equal running time will be compared. As discussed before, the hypervolume indicates the overall quality of the archive or population. All experiments regarding the optimization of PLS-MOLA and IPLS-MOLA will be applied on the (complete) Brazil case. Finally, several experiments are set up that will give more insight in how the performance of (I)PLS-MOLA compares to NSGA-II-MOLA. As NSGA-II is currently the most used algorithm in the MOLA domain, this will give an idea of the relative efficiency of PLS-MOLA for MOLA. These comparisons will also be done using the subcases, to gain more insight in the scalability as well.

In Chapter 8.3.2 to 8.3.7, eight test cases have been set up to examine and compare the algorithm components stated in the research questions of Chapter 1.3. Each test case consists of multiple algorithm 'runs', in which an algorithm is either executed for a certain number of iterations or running time. Together with each run, the parameter settings, what should be measured, and why this is relevant to a research question will be outlined. As a run is non-deterministic, each run will be executed 5 times. Executing each run more than 5 times is considered to be out of the scope of this research (as some runs take up to 2.5 hours), but could be done in future research to increase the reliability. The runs will be executed using the implementation of Chapter 7 and the Brazil case of Chapter 8.2 with the land-use map of Brazil of 2012 as input solution. The problem parameters (Chapter 7.4) that will be used, are shown in Table 8.7. Using these parameters, the problem corresponds to the problem definition described in Chapter 8.2.2. The number of types is set to 12, although type 1 and 2 are actually empty (as these were added to 12). Type 12 now includes all static cells, and is marked as static.

TABLE 8.7: Problem Parameter Setup

Id	Parameter	Value
1	Objectives	Compactness, Yield
2	Constraints	Allocation Ranges
3	Number of Types	12
4	Static Types	1, 2, 12
5	Number of Neighbors	4

Compactness and potential yield maximization are set as the two 'standard' objectives of the problem. For simplicity reasons, the choice has been made to start with the lowest number of objectives necessary to create a multi-objective problem, which is two. The effects of expanding the problem to three objectives are examined later in Chapter 8.3.7. Compactness has been chosen as the first objective to avoid clusters from becoming completely scattered. Potential yield optimization has been chosen as the second objective as the profit/economic viability is assumed to have a high influence on land-use change. In the test case of Chapter 8.3.7, the third objective of carbon stock maximization will be included as well.

The allocation ranges constraint requires a set of allowed allocation ranges as input. Research subquestion 1B questions the influence of these range sizes on the performance of the algorithm. Preferably, the range is as small as possible, as this allows the user to have more control. However, smaller ranges (or even a range of 0) are more likely to increase the average running time necessary to complete one iteration, as more constraint violations (and so reparations) are expected to occur. Therefore, the influence of the allocation range sizes on the running time to complete a certain number of iterations will be examined in the first case described in Chapter 8.3.2. Depending on these results, allocation ranges will be chosen for the remaining experiments. This choice will be based upon what size assumed to be a suitable trade-off between control and performance. Eventually, this choice is up to the end user, as different sizes also result in different problems that are being optimized. The area sizes of Table 8.3 for 2030 will be used as the required number of cells per type.

TABLE 8.8: PLS-MOLA Parameter Setup

Id	Parameter	Value
6	Selection Operators	s-r, s-cd
7	Search Operators	ls-krcm, ls-krbm, ls-krpm
8	Repair Operators	lr-krcm, lr-krbm
9	Search KT	4
10	Repair KC	10
11	Repair BT	True
12	Allow Repair	True
13	NDS	False
14	Max. Archive Size	∞
15	Exploration N	100
16	Exploration M	∞
17	Reparation N	∞
18	Reparation M	∞
20	PLS M	∞

Next up, each test case will examine the influence of a different (set of) parameter(s). In order to avoid having to redefine all PLS-MOLA parameter values for each case, a default set of parameters has been set up, as is shown in Table 8.8. These standard values will be used in all experiments where PLS-MOLA is applied, except for when mentioned differently. For the selection, search and repair operators, all operators have been added to decrease the influence of each operator individually. The KT value of the search operators has been set to 4, which is half of the maximum number of dynamic types that can compete a type selection tournament for Brazil (Chapter 8.3.4). For now, the Repair BT has been set to true and the Repair KC has been set to 10. These will both be further examined in Chapter 8.3.5. Next up, reparations are allowed during the optimization process, and the NDS criterion is set to false to do not make the acceptance criteria to strict. Furthermore, the archive size, Reparation N and both the Exploration and Reparation M values are set to infinity. Examining all of these parameters is out of the scope of this research, and by setting these to infinity they will no further influence the optimization process. The Exploration N value has been set to 100, and will be further examined in Chapter 8.3.4. Parameter 19 is left out, as this concerns the number of iterations of the algorithm. This number will be given with each experiment, except for when the algorithm is run for a certain amount of time; this means the number of iterations becomes irrelevant (infinite).

Finally, all algorithms will be run while allowing concurrency. As was already discussed in Chapter 7.5, parallelism does not affect the result of any algorithm, but only decreases the running time necessary for one iteration. Therefore, it should be rewarded if an algorithm supports parallel processing. All algorithms will be run while allowing multiple threads, as shown by the parameter setting in Table 8.9.

TABLE 8.9: Other Parameter Setup

Id	Parameter	Value
38	Asynchronous	True

8.3.2 Test Case I: Allocation Ranges

The first test case examines the influence of different allocation range sizes on the running time of the algorithm. The smaller the allocation range sizes, the closer the final solutions will be to the predictions of 2030 in terms of area sizes. However, a smaller allocation range also results in 'faster' violations of the bounds, resulting in more repair operations. Hence, research question 1B states whether and to what extend the range sizes influence the running time necessary to perform an equal number of iterations. In 8.10, six PLS-MOLA runs with different range sizes are shown. For now, the assumption is made that the user would at most want a range size of 100. In future experiments, larger range sizes can be experimented with as well. Again, each run will be executed 5 times. The range sizes will be applied to all five land-use types of Table 8.4 using the predicted area sizes of 2030. All other parameters of PLS-MOLA will be set according to the standards defined in Chapter 8.3.1.

TABLE 8.10: PLS-MOLA Runs with Range Size 0 - 100

Run	Range Size	PLS Iterations
1	0	1000
2	5	1000
3	10	1000
4	25	1000
5	50	1000
6	100	1000

Each run will be execute for a 1000 iterations, and the running time will be measured. This will provide insight in the relation between range size and running time. Note, that it is not valid to compare the quality of the resulting solutions (for example hypervolume), as different allocation ranges create different problem definitions. Higher range sizes could therefore allow for solutions with higher potential yields than solutions obeying smaller ranges. As so, only running times for completing the same number of iterations will be compared.

8.3.3 Test Case II: Selection Operators

The second case will examine both the performance and complexity of the two selection operators defined for PLS-MOLA; S-R and S-CD (Chapter 6.5). The first randomly selects a solution from the archive for exploration, whereas the second selects the solution with the highest crowding distance. S-R therefore costs less operations than S-CD, as it does not have to calculate the CD of each solution. On the other hand, S-CD is expected to provide more interesting solutions (with a less explored neighborhood), and could so result in a faster and/or better overall convergence. First of all, three PLS-MOLA runs have been defined to get insight in the overall optimization quality (performance) of each operator, as shown in Table 8.11. Each run will be executed 5 times, and run for 5 minutes. The hypervolume of the final archive will be measured, to indicate the optimization efficiency of the used operator(s). In run 9, both operators are used, as they might influence/complement each other. The other PLS-MOLA parameters are equal to the standards defined in Chapter 8.3.1.

TABLE 8.11: PLS-MOLA Runs with Different Selection Operators (Time)

Run	Selection Operators	Running Time
7	s-r	5 min
8	s-cd	5 min
9	s-r, s-cd	5 min

The previously defined three runs will examine the overall optimization efficiency of an operator. However, this does not give any insight in the complexity (costs) of the operators. Therefore, three more runs will be executed as shown in Table 8.12. The algorithms will be run for a certain number of iterations instead of a predefined running time. As so, we can measure the running time that the algorithm takes to complete these iterations. The running time will provide insight in the costs of the operators. The higher the running time, the higher the costs to run this operator. Eventually, the overall optimization efficiency examined previously in run 7 to 9 is a trade-off between the costs and the effectiveness of the operator.

TABLE 8.12: PLS-MOLA Runs with Different Selection Operators (Iterations)

Run	Selection Operators	PLS Iterations
10	s-r	1000
11	s-cd	1000
12	s-r, s-cd	1000

8.3.4 Test Case III: Search Operators

The third test case examines the performance and complexity of the three search operators defined for PLS-MOLA; LS-KRCM, LS-KRBM and LS-KRPM (Chapter 6.8). The operators are assumed to have different complexities (costs) and effectiveness when applied to MOLA problems. For example, the LS-KRPM operator changes complete patches, and thus requires a high number of operations. On the other hand, this might result in more qualitative solutions, paying off the extra operations. First of all, the performance (efficiency) will be examined in a similar manner as with the selection operators of Chapter 8.3.3. Seven runs will be executed as shown in Table 8.13. Each run uses a different (set of) search operators and will be executed for 5 minutes (5 times). Since search operators might influence and/or complement each other, all combinations will be examined. Of each run, the hypervolume of the final archive will be measured, which indicates the overall optimization quality of the used (set of) search operators(s). All other PLS-MOLA parameters will be set according to the standards of Chapter 8.3.1.

TABLE 8.13: PLS-MOLA Runs with Different Search Operators (Time)

Run	Search Operators	Running Time
13	ls-krcm	5 min
14	ls-krbm	5 min
15	ls-krpm	5 min
16	ls-krcm, ls-krbm	5 min
17	ls-krcm, ls-krpm	5 min
18	ls-krbm, ls-krpm	5 min
19	ls-krcm, ls-krbm, ls-krpm	5 min

Again, the results of run 13 to 19 only focus on the overall optimization efficiency. As so, the runs of Table 8.14 have been added to get insight in the complexity as well. These will all be run for a 1000 iterations (5 times), of which the total running time necessary to complete these iterations will be measured. Similar to the selection operators, this will provide insight in the complexity (costs) of each operator. Again, the overall optimization quality resulting from run 13 to 19 is a trade-offs between the costs and effectiveness of the operators. Run 20 to 26 will provide insight in this trade-off.

TABLE 8.14: PLS-MOLA Runs with Different Search Operators (Iterations)

Run	Search Operators	PLS Iterations
20	ls-krcm	1000
21	ls-krbm	1000
22	ls-krpm	1000
23	ls-krcm, ls-krbm	1000
24	ls-krcm, ls-krpm	1000
25	ls-krbm, ls-krpm	1000
26	ls-krcm, ls-krbm, ls-krpm	1000

Except for the search operators being used, the search KT parameter also influences the search process. The search KT parameter, as described earlier in Chapter 6.8, states the number of types participating in the type selection tournament of the operators. The maximum number of types competing in this tournament, is the number of dynamic types minus one (as the current type of the cell may not participate). This results in a maximum tournament size of 9 - 1 = 8 for the Brazil case. A KT of 1 will hereby result in a random type being selected, whereas a KT of 8 results in a greedy selection of the best type. A higher KT is therefore expected to result in faster increasing objective values. However, it will also increase the running time necessary for an iteration as more operations need to be executed. In order to answer whether this trade-off pays off, 8 PLS-MOLA runs will be performed (5 times each) with the search KT ranging from 1 to 8. The runs are shown in Table 8.15, and will all be run for 5 min. The final archive hypervolumes will be measured, to indicate the optimization effectiveness. All other parameters are set to the prediscussed standards of Chapter 8.3.1.

TABLE 8.15: PLS-MOLA Runs with Search KT 0 - 8

Run	Search KT	Running Time
27	1	5 min
28	2	5 min
29	3	5 min
30	4	5 min
31	5	5 min
32	6	5 min
33	7	5 min
34	8	5 min

Finally, the search process is influenced by the Exploration N parameter. This parameter states the number of neighborhood solutions to be explored in each exploration iteration. A high value of N will result in a more extensive neighborhood exploration, at the cost of a higher running time for one iteration. As such, different exploration N values ranging from 10 to 1000 will be examined, as shown in Table 8.16. In future research, exploration N values of above 1000 might be examined as well, in case deemed interesting. For now, this was out of the scope of this research. Each run will be executed 5 times and run for 5 minutes. The final archive hypervolume will be measured, to indicate the solution quality. Note, that the 'optimal' Exploration N value might differ over the number of iterations. For example, it could pay off to explore more elaborately in later stages, when finding improvement gets more difficult. However, examining this is out of the scope of this current research.

TABLE 8.16: PLS-MOLA Runs with Exploration N 10 - 1000

Run	Exploration N	Running Time
35	10	5 min
36	25	5 min
37	50	5 min
38	100	5 min
39	250	5 min
40	500	5 min
41	1000	5 min

8.3.5 Test Case IV: Reparation Operators

The fourth case examines the two reparation operators for PLS-MOLA defined in Chapter 6.10: LR-KRCM and LR-KRBM. The reparation operators affect both the quality of the repaired solutions and the reparation time necessary. The operators are examined in the same manner as the selection and search operators of Chapter 8.3.3 and 8.3.4. First, three runs will be executed with different (sets of) repair operator(s), as shown in Table 8.17. Each runs will be executed 5 times, for 5 minutes, and the resulting archive hypervolumes will be measures. This will give insight in the effectiveness of the reparation operators in terms of final solution quality. All remaining PLS-MOLA parameters will be set as discussed in Chapter 8.3.1.

TABLE 8.17: PLS-MOLA Runs with different Repair Operators (Time)

Run	Repair Operators	Running Time
42	lr-krcm	5 min
43	lr-krbm	5 min
44	lr-krcm, lr-krbm	5 min

Again, in order to gain more insight in the complexity, 3 more runs will be executed. These are shown in Table 8.18 and will all be run for 1000 iterations (5 times). Of each run, the total running time will be measured, indicating the costs of the operator(s) being used. Similar to the previous operators, this will provide insight in the trade-off between costs and effectiveness.

TABLE 8.18: PLS-MOLA Runs with different Repair Operators (Iterations)

Run	Repair Operators	PLS Iterations
45	lr-krcm	1000
46	lr-krbm	1000
47	lr-krcm, lr-krbm	1000

As discussed in Chapter 6.10, there are two parameters that can be used to adjust the behavior of the repair operators. The first parameter is the repair KC parameter, which states the number of (boundary) cells competing in the operator its tournament. A higher KC value will result in a more greedy hill climbing approach, but also increases the number of operations necessary (costs). As so, a higher KC value is expected to deliver higher quality solutions, but also require more reparation time. As shown in Table 8.19, 6 runs will be done given different repair KC values ranging from 1 (the minimum) to 100. In future research, KC values of above 100 can be examined as well, in case deemed interesting. For now, this is out of the scope of this research. Instead of executing these runs for either a certain time or number of iterations, run 48 to 53 will terminate after the initial solution has been repaired. The reason for this, is that it is unknown to what extent high KC values will increase the reparation time necessary. As so, a predefined number of iterations could take extremely long, whereas a certain timespan (for example 5 minutes) might not be enough to repair even one solution. Repairing the initial solution only, will prevent both problems from happening. The final hypervolume and time necessary to repair the initial solution will be measured. This will give insight in both the effectiveness (reparation quality) and costs (reparation time). All other parameters will be set according to the standards defined earlier, meaning that both LR-KRCM and LR-KRBM will be selected.

TABLE 8.19: PLS-MOLA Runs with Repair KC 1 - 100

Run	Repair KC	PLS Iterations
48	1	1
49	10	1
50	50	1
51	100	1
52	250	1
53	500	1

The second parameter for setting up the repair operators is the Repair BT parameter. If set to true, the best type (considering one random objective) will be chosen for each cell competing in the tournament. Otherwise, a random type will be selected. As shown in Table 8.20, the PLS-MOLA algorithm will be run with Repair BT both set to true and false. All other parameters will be equal to the standard settings discussed earlier. With Repair BT set to true, the costs of the repair operators are expected to be higher. On the other hand, the quality of the repaired solutions is expected to be higher as well. Each run will be executed 5 times for 5 minutes, and the final archive hypervolumes will be measured.

TABLE 8.20: PLS-MOLA Runs with and without Repair BT

Run	Repair BT	Running Time
54	No	5 min
55	Yes	5 min

Finally, it is optional to not allow PLS-MOLA to repair solutions during the optimization process. This is determined by the Allow Repair (AR) parameter discussed in Chapter 7.4. If set to false, invalid solutions resulting from a search operator are discarded, and a new search is performed until a valid one is found. The initial solution will be repaired at all times. Two tests will be executed (5 times each) as shown in Table 8.21. Both runs will be executed for 5 minutes and the final archive hypervolume will be measured to indicate the effectiveness. This will provide insight in whether repairing is worth the number of operations necessary. If not, immediately generating new solutions could be more beneficial, and result in higher hypervolumes.

TABLE 8.21: PLS-MOLA Runs with and without Allow Repair (Time)

Run	Allow Repair	Running Time
56	No	5 min
57	Yes	5 min

8.3.6 Test Case V: Acceptance Criteria

In the fifth case, two parameters regarding the acceptance criteria for the archive will be examined. First of all, the NDS parameter, which if set to true results in solutions only being accepted to the archive in case they dominate the current solution being explored. If set to false, solutions can also be accepted in case they do not dominate the current solution being explored (and of course are non-dominated by all solutions in the archive). Set to false, more solutions will be accepted (and discarded again if dominated), but the average quality of these solutions is expected to be lower. The question is, whether the increase in quality is worth the decrease in the number of solutions. A lower number of solutions, might eventually lead in the algorithm terminating more easily in later stages. This is caused by the fact that it gets more difficult to find improvements in later stages, in which case it might also be beneficial to accept and explore these less qualitative solutions. Therefore, both the influence on the hypervolume and the number of solutions in the archive will be examined. As shown in Table 8.22, two runs will be executed

(5 times each) for 5 minutes, with NDS set to either true or false. The remaining PLS settings are set according to the standards of Chapter 8.3.1. Both the hypervolume and size of the final archive will be measured.

TABLE 8.22: PLS-MOLA Runs with and without NDS Criterion

Run	NDS	Running Time
58	No	5 min
59	Yes	5 min

The second parameter related to the archive acceptance criteria, is the maximum archive size. In case the maximum archive size is exceeded, the solution with the lowest crowding distance will be removed. A lower archive size will result in less solutions in the archive, decreasing the number of operations necessary to check for dominance. This again decreases the operational costs and memory needed. However, the question is how this will influence the effectiveness and quality of the resulting archive. Also, similar to with NDS, a decrease in the archive size could result in an earlier termination of the algorithm in later stages. In the standard settings of Chapter 8.3.1, the size was set to infinity, accepting any number of solutions in the archive simultaneously. As shown in Table 8.23, seven runs will be executed (5 times each) with the archive size ranging from 5 to 100. As the standard Exploration N was set to 100, this has also been selected as the largest maximum archive size to be explored in this research. Theoretically, the archive size can become larger than 100 with an Exploration N of 100, but especially in the first 5 minutes of the algorithm where improvements are relatively easy to find, this is assumed to be unlikely. Of each run, the hypervolume of the final archive will be measured.

TABLE 8.23: PLS-MOLA Runs with Max. Archive Size 5 - 100

Run	Max. Archive Size	Running Time
60	5	5 min
61	10	5 min
62	15	5 min
63	20	5 min
64	25	5 min
65	50	5 min
66	100	5 min

8.3.7 Test Case VI: Objectives

In this test case, the effects of optimizing a third objective will be examined. In RQ2, the relation between the number of objectives and the efficiency of PLS-MOLA is being questioned. So far, we have been optimizing two objectives; compactness and potential yield maximization. In Chapter 8.2.3.3, a third objective has been introduced; carbon stock maximization. In this test case, the effectiveness and costs of optimizing all three objectives simultaneously will be compared to when optimizing compactness and yield only. First, two runs will be performed (5 times each) as shown in Table 8.24. Both runs will be executed for 1000 iterations and both the running time and number of solutions

in the final archive will be measured. The running time will give insight in the increase in computational costs by the addition of an objective. The number of solutions will confirm whether the addition of an objective affects the archive size. Expected, is that a third objective will result in a larger archive size, as a solution is dominated less quickly. A larger archive size, could lead to a higher demand in memory and possibly increase the processing time as more operations are required (for example to check dominance). All other parameters are set according to the PLS-MOLA standards discussed in Chapter 8.3.1.

TABLE 8.24: PLS-MOLA Runs with Different Objectives

Run	Objectives	PLS Iterations
67	Compactness, Yield	1000
68	Compactness, Yield, Carbon	1000

Secondly, two runs will be executed to examine the influence of adding more objectives on the resulting solution quality (hypervolume) of the archive, as shown in Table 8.25. Each run will be executed 5 times for 5 minutes, and the hypervolume of only the compactness and yield objectives will be measured. For run 70, this means that the objective values of carbon are neglected for calculating the hypervolume. By doing so, more insight will be given in the influence of the extra objective on the optimization effectiveness of the objectives individually. Expected, is that the compactness and yield objective values will be lower in Run 70, as the local search operations will now (approximately 1/3 of the time) focus on the carbon stock objective as well.

TABLE 8.25: PLS-MOLA Runs with Different Objectives

Run	Objectives	Running Time
69	Compactness, Yield	5 min
70	Compactness, Yield, Carbon	5 min

8.3.8 Test Case VII: IPLS Optimization

In the seventh test case, the focus will be on how the PLS algorithm can efficiently be processed into a global IPLS algorithm for MOLA. The IPLS algorithm iteratively calls the PLS algorithm. Therefore, the results of Test Case I to VI are also relevant for IPLS, as the optimal parameter settings to call the PLS algorithm can be based upon these results. In Test Case VIII, the IPLS algorithm will eventually be run for 2.5 hours to examine and compare its performance with NSGA-II. Larger running times are considered to be out of the scope of this research. However, this does require the PLS algorithm to (on average) reach a local optimum within the timespan of 2.5 hours for IPLS to be applicable. The PLS algorithm is said to have found such a local optimum, when it is not able to find a new neighboring solution before the exploration termination criteria are met and so terminates. In order to determine for which cases PLS (on average) terminates within 2.5 hours, we will first run the PLS-MOLA algorithm on all three cases; the large Brazil case, the medium sized Centre West case, and eventually the small Sul Goiano case. View Chapter 8.2 for more details on these cases. On each of these cases, the algorithm will run for at most 2.5 hours using the parameters shown in Table

8.26. As can be seen, all parameters (6 to 15) will be based on the results of Test Case II to V. The parameters being chosen will be the parameters that resulted in the best (trade-off between) running time and archive hypervolume. In order to be able to visualize the archive of PLS-MOLA over time, the archive will be stored each 15 minutes, 10 times in a row. The algorithm will run for a most 2.5 hours, unless terminated earlier, for 5 times per problem case. The average, minimum and maximum yield and compactness of the final archive will be stored as well. These will optionally be used in Test Case VIII to compare with the NSGA-II algorithm, as will be explained later.

TABLE 8.26: PLS-MOLA Parameters Settings

Id	Parameter	Value
6	Selection Operators	*Results Test Case II*
7	Search Operators	*Results Test Case III*
8	Repair Operators	*Results Test Case IV*
9	Search KT	*Results Test Case III*
10	Repair KC	*Results Test Case IV*
11	Repair BT	*Results Test Case IV*
12	Allow Repair	*Results Test Case IV*
13	NDS	*Results Test Case V*
14	Max. Archive Size	*Results Test Case V*
15	Exploration N	*Results Test Case III*

After having determined what cases IPLS will be ran on in this research (namely the ones for which PLS terminates in less than 2.5 hours), effective perturbation sizes will have to be determined. The perturbation size states the percentage of the solution (dynamic cells) that will be randomized. If the perturbation size is too small, the solution will not be apply to 'leave' its current local optimum. If the perturbation size is too big, the solution will lose a lot up to all of its currently well-optimized parts. As so, three runs will be performed as shown in Table 8.27. The runs implement a perturbation size of either 25%, 50% or 75%. Note, that a size of 0% or 100% would not make sense; the first does not result in any perturbation at all whereas the second creates a completely randomized solution (losing all valuable information that was present). Each run will be executed 5 times on each case for which the PLS algorithm found an optimum within 2.5 hours. For each run, the final archive hypervolume will be measured. The perturbation size resulting in the highest hypervolume will be preferred. The remaining parameters are equal to the standard parameters stated in Chapter 8.3.1.

TABLE 8.27: IPLS-MOLA Runs with Perturbation Size 10000 - 1000000

Run	Perturbation Size	IPLS Iterations	PLS Iterations
71	25	10	*Results Test Case VII*
72	50	10	*Results Test Case VII*
73	75	10	*Results Test Case VII*

8.3.9 Test Case VIII: NSGA-II Comparison

In the final test case, the performance of NSGA-II will be compared to either PLS or IPLS. For all cases where the PLS algorithm did not terminate within 2.5 hours, the NSGA-II performance will be compared to PLS. The performance of the PLS algorithm for these cases was already obtained in Test Case VII. For the cases where PLS did terminate on time, and IPLS could be applied, the IPLS algorithm will be run using the parameters shown in Table 8.28. The perturbation size will be set according to the results of run 71 to 73 of Test Case VII for that specific case. The algorithm will be run for 2.5 hours, and the archive will be stored each 15 minutes to get insight in the convergence of the algorithm over time. The average, minimum and maximum yield and compactness value of the final archive will be stored, and eventually be compare to NSGA-II. Comparing the statistics of each objective individually, gives more information than comparing hypervolume solely. One algorithm could be better at optimizing one objective than the other, which is not visible when looking at hypervolume only.

TABLE 8.28: IPLS-MOLA Parameters Settings

Id	Parameter	Type
21	Perturbation Size	*Results Run 71 to 73*

The NSGA-II-MOLA algorithm will be run on all three cases (Brazil, Centre West and Sol Goiano) for 2.5 hours, with the parameter values shown in Table 8.29. Since it is out of the scope of this research to optimize all parameters of this algorithm, values have been chosen that are expected to bring the best trade-off between costs and effectiveness. First of all, both crossover operators C-XTD and C-XBC have been chosen, to reduce the influence of one individual crossover operator. The mutation (search) operators and repair operators will be set according to Test Case III and IV, assuming that in case these work well for PLS-MOLA, these will also be effective for NSGA-II-MOLA. A population size of 100 was chosen, as a suitable trade-off between convergence quality and memory requirements. A larger population size could lead to a more effective optimization process due to the possibility to store more (different) genes. However, this comes at the cost of higher memory storage requirements (especially in later stages, as we are storing differences with the initial map). As the algorithm is currently ran on a personal computer, larger populations sizes might therefore cause memory problems. In future research, it is advised to experiment with larger population sizes and further examine what size is most effective (Chapter 11). The initial population is set to exist for 50% out of randomly generated solutions, and 50% out of mutations of the current solution. As the ideal ratio is unknown, an equal share has been devoted to both. For the same reason, the mutated solutions mutate 50% of the current solution. The Mutation KT, Repair KC and Repair BT are set based upon the results of Test Case III and IV, assuming these will also hold for NSGA-II-MOLA. The Mutation N is set to 100, meaning each solution is mutated (once per generation) by applying a local search operator for 100 times. This was considered as a suitable amount of local search within the NSGA-II algorithm. In future research, the amount of local search can be increased or decreased, to view whether this improves the optimization efficiency. Finally, all remaining termination criteria are set to infinity to not limit the reparations, as there is currently no knowledge about when and whether

this would be profitable. Again note, that all of these values are based upon personal assumptions about what will be effective and will most likely not be optimal. For now, researching these values is out of the scope of this research, as will be discussed in Chapter 11. Similar to with IPLS-MOLA, the algorithm will be run for a total of 5 times and its population will be stored every 15 minutes. Also, the same statistics of the final population will be stored as was done with the archive of PLS and IPLS.

TABLE 8.29: NSGA-II-MOLA Parameters Settings

Id	Parameter	Value
23	Crossover Operators	c-xtd, c-xbc
24	Mutation Operators	*Results Test Case III*
25	Repair Operators	*Results Test Case IV*
26	Population Size	100
27	Init. Random Sol. (%)	50
28	Init. Mut. Size (%)	50
29	Mutation N	100
30	Mutation KT	*Results Test Case III*
31	Repair KC	*Results Test Case IV*
32	Repair BT	*Results Test Case IV*
33	XBC N	100
34	XBC M	∞
35	Reparation N	∞
36	Reparation M	∞

The statistics of the resulting population of NSGA-II will eventually be compared to the statistics of the resulting archives of PLS or IPLS. Hereby, a comparison will be done for each of the objectives separately, to see which algorithm performs best for which objective(s) in what case(s). As so, for each objective, a two-tailed independent sample t-test will be performed on the objective values obtained by different algorithm. Again, note that comparing the objectives separately gives a more extended view than comparing hypervolumes only, as the hypervolume is (easily said) a combination of both. This will be done for each of the three cases, and allows us to view what algorithm optimizes what objective(s) best in what case.

8.4 Materials

All experiments will be executed on a Microsoft Surface Pro 7 (Microsoft, 2019a). The hardware specifications deemed relevant for running the optimization algorithm are summarized in Table 8.30. The clock rate, number of cores, threads and execution units are related to the processor. The last specification describes the RAM memory (both size and type) being used.

TABLE 8.30: Hardware Specifications

Processor	Intel® Core™ i7-1065G7 (10th Generation)
Clock Rate	1.30 GHz up to 3.90 GHz
Number of Cores	4
Number of Threads	8
Execution Units	64
RAM Memory	16GB (3733 MHz LPDDR4X)

Finally, the software specifications deemed relevant for running the optimization algorithm are shown in Table 8.31. These are the integrated development environment (IDE) used to run the algorithm, the development framework that is it build upon and the operating system running the IDE.

TABLE 8.31: Software Specifications

Development Environment	Visual Studio Professional 2019 (Version 16.5.4)
Development Framework	.NET Framework (Version 4.8.0375)
Operating System	Windows 10 Pro (Build 18363)

Chapter 9

Results & Discussion

9.1 General

In this chapter, the results of the conducted experiments will be outlined, interpreted and discussed. The results are hereby grouped per research question being answered. At first, all results regarding the optimization of PLS-MOLA (RQ 1) will be discussed in Chapter 9.2. This mostly concerns what operators and procedures have proven to be most effective for the Brazil case. Secondly, the scalability of PLS-MOLA concerning the number of objectives being optimized (RQ 2) will be handled in Chapter 9.3. Thirdly, optimization of IPLS-MOLA (RQ 3) will be outlined in 9.4. This will be done for the Brazil, Centre West and Sul Goiano case. Finally, the performance of the (I)PLS-MOLA algorithm will be compared to the NSGA-II-MOLA algorithm (RQ 4). Again, this will be done for all three test cases. For some experiments, the parameter settings were based upon results of previous experiments. These settings will be justified and explained as well. All raw statistics of the experimentation results can be found in Appendix G and H.

9.2 RQ I: PLS Optimization

The first research question states "How can PLS most efficiently be applied to MOLA?". In order to answer this question, six subquestions were set up to examine influence of the selection, exploration and reparation strategies, acceptance criteria, data structure and allocation range sizes on the efficiency of PLS-MOLA. In this chapter, each subquestion will be answered using the results obtained by the experiments. Only the first subquestion, regarding an efficient data structure, will be answered with reasoning (not empirically). Together, the outcomes of all six subquestion answer the first research question of how an efficient PLS algorithm for MOLA can be built.

9.2.1 Data Structure

The first subquestion of RQ 1 states "What is an efficient data structure for storing MOLA solutions?". The most straightforward data structure would naively store the land-use type of each cell of the map. However, as the Brazil case is a relatively large case and new solutions are continuously being created and stored during the PLS optimization process, this could result a high memory demands. In order to decrease these demands, a data structure was designed to store MOLA solutions more efficiently. This resulted in the design of an 'initial' and 'derived' solution, as was shown in Figure 7.2. An initial solution is stored in the naive manner, taking note of the type of each cell. A derived solution on the other hand, stores a reference to an initial solution and only takes note of the adjusted cell types. As the number of adjusted

cells is always below or equal to the total number of cells, this data structure will require less or equal amount memory space than an initial solution. The PLS algorithm could now store the input map of the case study as an initial solution, and create derived solutions referencing to this map during the optimization process. As the type of some cells never change, these are now stored only once, thus decreasing the redundancy. The more adjustments are being made to the initial map, the higher the memory required for a derived solution will be.

No experiments were conducted to measure the memory requirements during the optimization of PLS and/or NSGA-II. As so, the reduction in memory requirements is assumed to be occur due to the reasoning above. In Chapter 11, more research in this direction is advised to examine the actual memory requirements and further improve upon the current structure being used.

9.2.2 Allocation Range Size

The second subquestion of RQ 1 states "What is the influence of the allocation range size on the running time?". The allocation ranges, which are part of the most common MOLA constraint, state to how many cells each land-use type should be allocated (Chapter 3.2). It is expected that the larger the size of the allowed range, the less violations occur. This will decrease the running time necessary to perform the same number of iterations (as less 'reparations' are required), but also decreases the 'control' of the user on the final solutions. This subquestion examines the relationship between range size and running time. As such, Test Case I of Chapter 8.3.2 was set up and executed, testing the influence of different range sizes on the running time. The results of this experiment, concerning run 1 to 6, are shown in Figure 9.1 (Table G.1). As can be seen, the lower the range size, the higher the running time of the algorithm necessary to complete 1000 iterations on the Brazil case. This was expected, as more reparations will be required. According to Figure 9.1, running time decreases quickly when increasing the range size from 0 to 10. When further increasing the range size from 10 to 100, a slower decrease in running time occurs. The running time with a range size of 10 (564659.2ms) is even sightly lower than with a range size of 25 (565019.2ms), due to an outlier in the latter. For this research, a range size of 10 was considered to be a suitable trade-off between accuracy and running time. This results in a deviation of at most 5 cells per land-use type from the estimations, which is a relatively low number of cells compared to the total number of dynamic cells of Brazil. Note, that in the end it is up to the user to decide upon what range size is suitable given the situation of the MOLA problem, as a different size results in a different problem definition. For now, a range size of 10 has been chosen for the remaining of the experiments.

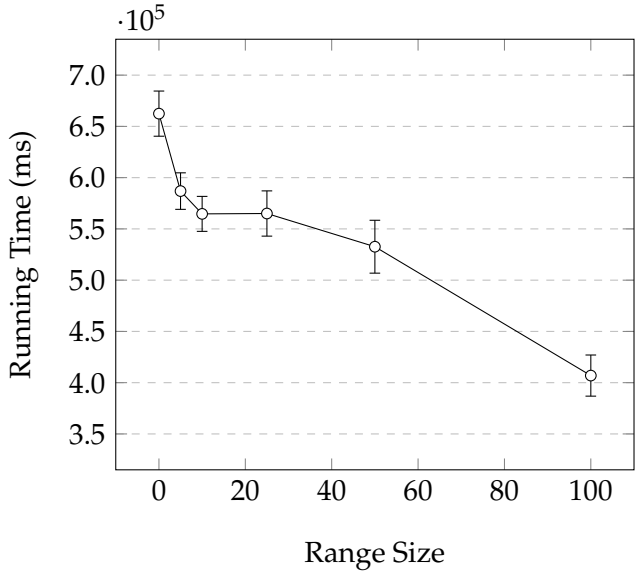


FIGURE 9.1: Range Size versus Average Running Time for PLS-MOLA (1000 Iterations, N = 5)

9.2.3 Selection Strategy

The third subquestion of RQ 1 states "What is an efficient strategy for selecting the next solution to be explored?". The selection procedure of the PLS algorithm, decides upon the next solution of the archive that will be explored. The selection strategy exists out of a selection procedure and one or more selection operators. The selection procedure chooses what selection operator will be used in the current iteration. The selection operator decides, given the current archive, which solution its neighborhood will be explored next. Due to the scope of this project, only one basic selection procedure was implemented that selects a selection operator at random (Chapter 6.4). As for the selection operators, the S-R and S-CD operators have been examined. The S-R operator selects the next solution at random, whereas the S-CD operator selects the solution with the highest crowding distance. Both are explained in more detail in Chapter 6.5. In Figure 9.2, the results of Run 7 to 9 are shown (Appendix Table G.2). In these runs, PLS was run with either one or both of the operators on Brazil for 5 min, and the final hypervolumes were measured. As can be seen, the S-CD operator results in a slightly higher hypervolume (3.70380×10^{14}) than the S-R operator (3.69318×10^{14}). When performing a two-tailed independent sample t-test between the hypervolumes of S-R and S-CD, this results in a p-value of 0.083127. As this is above 0.05, the operators do not result in significantly different hypervolumes. Although the difference is not significant, the S-CD operator will be used for the final algorithm comparisons of Test Case VIII as it performed slightly better.

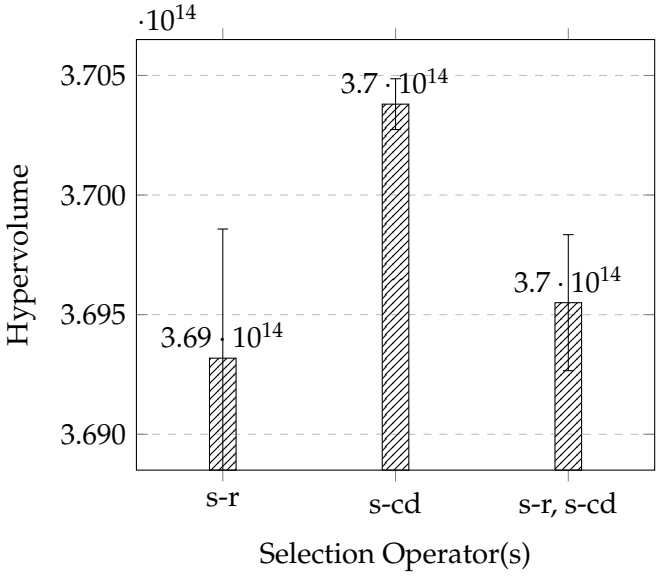


FIGURE 9.2: Average Hypervolume per Selection Operator(s) for PLS-MOLA (5 Min, N = 5)

In order to gain more insight in the complexity of the selection operators, run 10 to 12 were executed. The average running time to complete 1000 iterations on the Brazil case are shown in Table G.3 of Appendix G. As expected, the running time is considerably higher for the S-CD operator (5.61×10^5ms) compared to the S-R operator (5.06×10^5ms). The S-CD simply requires the calculation of the crowding distance of each solution, when a new solution is to be selected. However, despite the increase in efficiency (as the hypervolume was higher after the same running time), this did not lead to a significant improvement in efficiency, as was shown previously.

To conclude, no significant differences have been detected between the hypervolumes resulting from the different selection operators. As the usage of S-CD delivered a slightly higher hypervolume, this operator will be used when comparing the performance of (I)PLS-MOLA in Test Case VIII.

9.2.4 Exploration Strategy

The fourth subquestion of RQ 1 states "What is an efficient strategy for exploring the neighborhood of a solution?". The exploration strategy exists out of one or more exploration operators and an exploration procedure. The exploration operators are responsible for finding new neighbor solutions. The exploration procedure, as was discussed in Chapter 6.6, repeatedly applies (randomly selected) exploration operators to explore a neighborhood until the termination criteria are met. In this research, the efficiency of the LS-KRCM, LS-KRBM and LS-KRPM operators of Chapter 6.8 have been examined. Also, experiments regarding different KT values, influencing the working of these operators (Chapter 6.8), have been conducted. Finally, the Exploration N parameter of the exploration procedure was examined, which states the number of neighborhood solutions being explored during each iteration.

At first the exploration (search) operators were examined in run 13 to 19. The findings of these runs can be found in Figure 9.3 (Table G.4) and concern the average hypervolume obtained after 5 minutes on the Brazil case. Each run,

different (combinations of) search operators were used. Usage of only the LS-KRBM operator resulted in the highest hypervolume (3.69651×10^{14}ms), whereas a combination of all three operators resulted in the second best score (3.69095×10^{14}ms). A two-tailed independent sample t-test concerning the hypervolumes of both runs resulted in a p-value of 0.139405. As this is above 0.05, the operators are considered not to be significantly different in terms of efficiency. In order to decide upon what operators are preferred for the final comparison in Test Case VIII, we will therefore look at the expected diversity of the solutions being created. The combination of all operators, is hereby expected to (more easily) create a more diverse set of solutions. Also, the LS-KRBM is especially focusing on the compactness objective, and might therefore perform less good when this objective is not included. Therefore, despite of the slightly higher hypervolume of LS-KRBM, the preference will go to using all three operators in Test Case VIII.

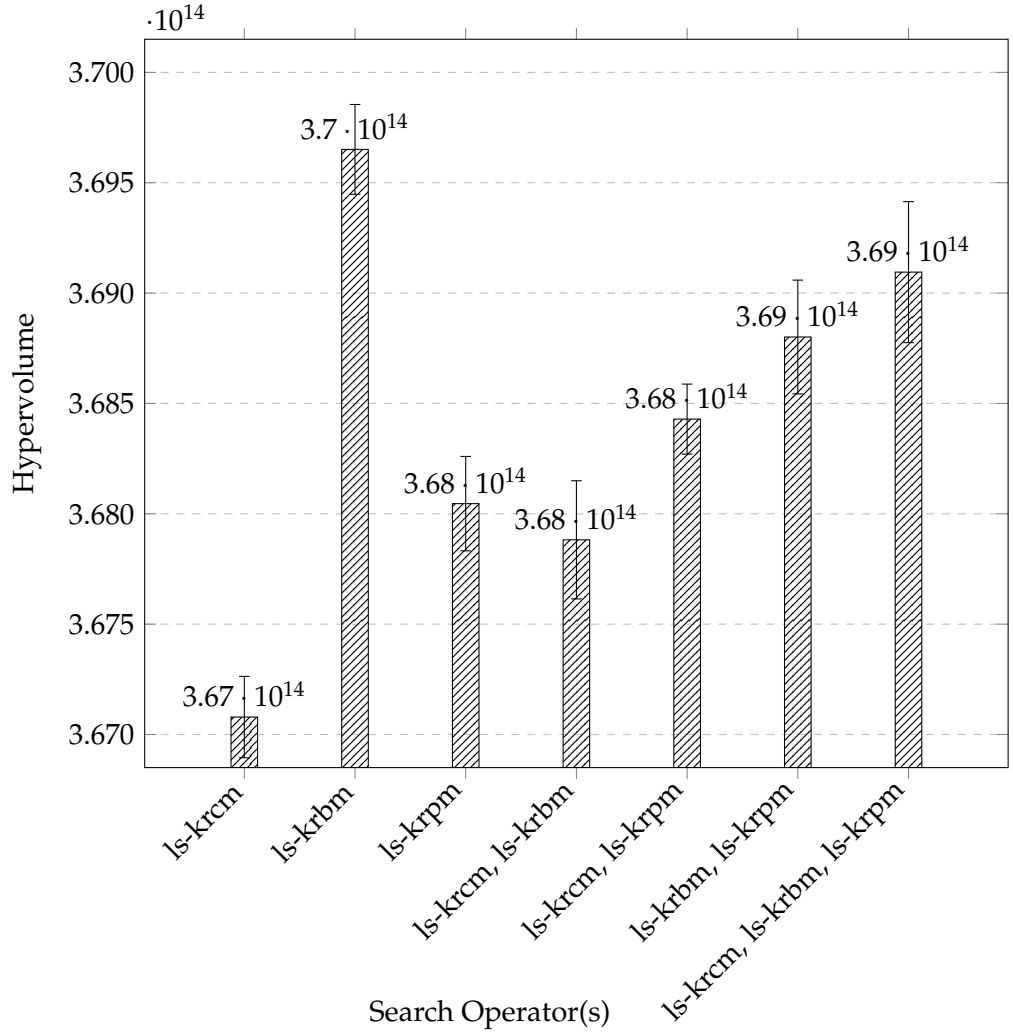


FIGURE 9.3: Average Hypervolume per Search Operator(s) for PLS-MOLA (5 Min, N = 5)

To gain more insight in the complexity of the operators, run 20 to 26 were executed. The results are shown in Table G.5 of Appendix G and show the running times necessary to complete 1000 iterations on the Brazil case for all combinations of search operators. As expected, the LS-KRPM results in the highest running time (1.24×10^6ms) compared to the LS-KRCM (2.46×10^5ms)

and LS-KRBM (1.61×10^5ms) operators. This is caused due to the fact that this operator alters seven cells at once, instead of only one. However, applying the LS-KRPM operator once seems to bring a bigger increase in hypervolume than applying the LS-KRCM or LR-KRBM once, as it is still able to reach comparable (slightly lower) hypervolumes after 5 minutes.

One of the parameters used to influence the behavior of each search operator, is the KT value (Chapter 6.8). The KT value states the number of types that are being compared, of which the one resulting in the best change in objective value is chosen. The results of run 27 to 34 are shown in Figure 9.4 (Table G.6), and indicate the average resulting hypervolume on the Brazil case when run for 5 minutes, using different KT values. A KT value of 4 results in the lowest hypervolume ($3.68667 \times 10^1$4ms) and a KT of 8 (the maximum, view Chapter 8.3.4) results in the highest hypervolume ($3.69256 \times 10^1$4ms). A KT of 1 has the second-to-highest hypervolume ($3.69158 \times 10^1$4ms). The fact that a KT of 4 performs the worst, is caused by the amount of random number generations necessary per operation. When randomly selecting 4 types, 4 random number generations are required. When randomly selecting 3 or 5 types, only 3 random number generations are required, and so on. As random number generations can be costly, this result in the algorithm being able to execute less iterations, which can have a negative effect on the final hypervolume. The fact that a KT of 8 performs slightly better than a KT of 1, indicates that comparing more types positively influences the resulting solution quality. Although comparing more types asks for more operations, it is apparently a positive trade given the increase in solution quality that is obtained. When performing a two-tailed independent sample t-test between the results of KT = 4 and KT = 8, this results in a p-value of 0.053596. As this is above 0.05, no significant difference is proven. Comparing KT = 1 to KT = 8 results in a p-value of 0.779688, again showing no significant difference. In the final comparison of Test Case VIII, a KT value of 8 will be used due to the slightly higher hypervolumes obtained.

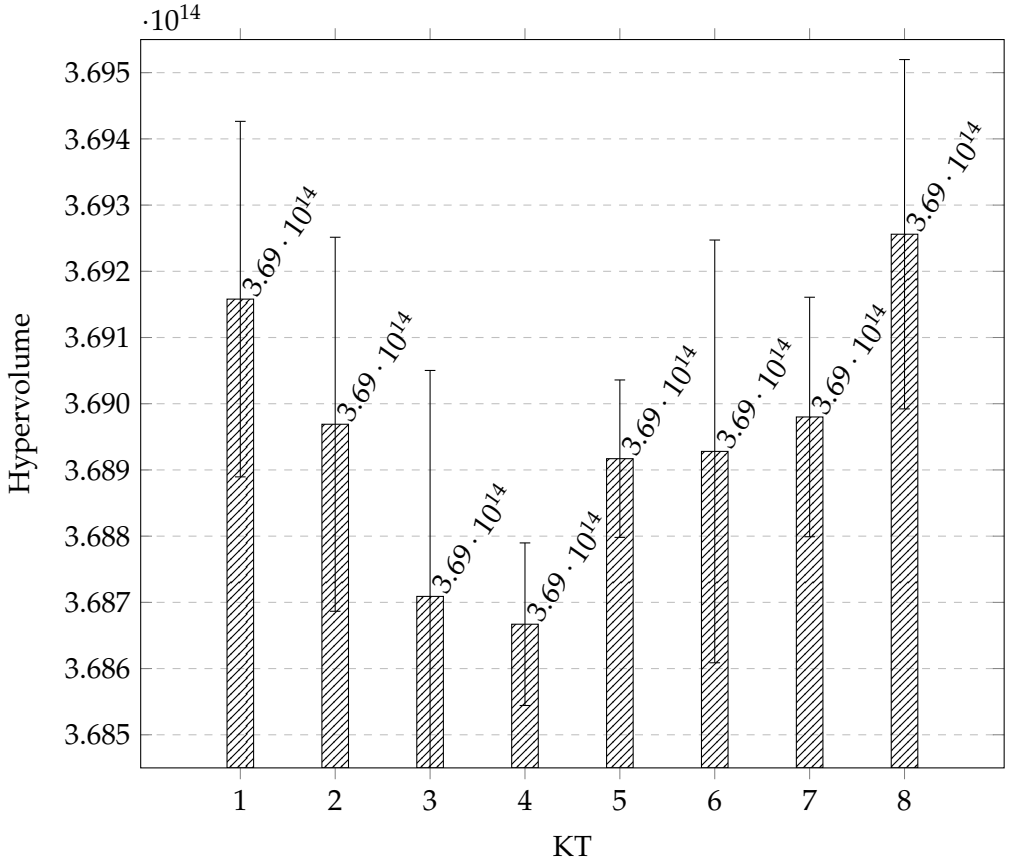


FIGURE 9.4: KT versus Average Hypervolume for PLS-MOLA (5 Min, N = 5)

Finally, the Exploration N parameter was examined in run 35 to 41 (Table G.7). This parameter indicates the number of neighborhood solutions being generated (explored) per neighborhood exploration. Figure 9.5 shows the average hypervolume obtained when ran on the Brazil case for different Exploration N values. As can be seen, a lower Exploration N value results in a higher average hypervolume. This can be explained by the fact that during the first 5 minutes, there are a lot of improvements possible for each solution. As so, it is almost certain improvements will be found fast, and so quickly moving on to the next iteration will speed up the optimization process. However, during later iterations, finding improvements can be more difficult. Low Exploration N values will be insufficient to find further improvements in time, which could terminate the algorithm early. Unfortunately, this is not the case within the 5 minutes being tested, and so longer tests should be performed to examine the optimal Exploration N value. For now, an Exploration N value of 10 has been used in future experiments. As shown in future experiments (see Chapter 9.4), the PLS-MOLA algorithm is able to run (at least) 2.5 hours using an Exploration N value of 10 on the Brazil case and around 1 hour on the Sul Goiano case. This is sufficient for the future experiments to be conducted successfully. In future research, experiments with longer running times should be conducted to examine the influence of the Exploration N value on the hypervolume and/or termination time in later stages more extensively.

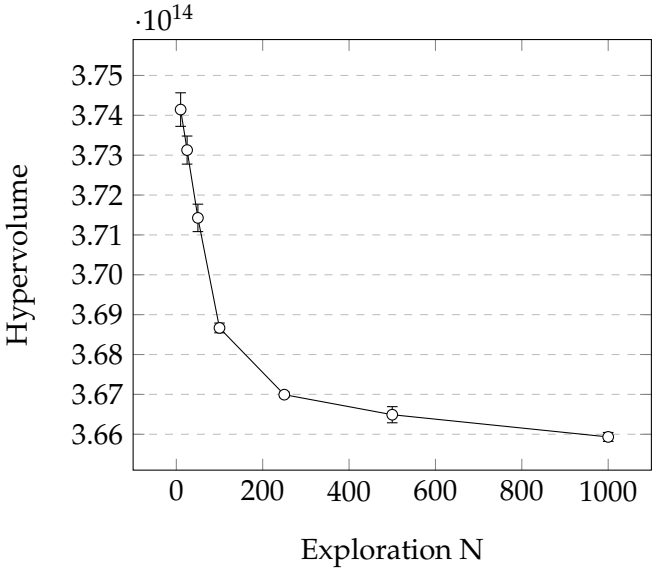


FIGURE 9.5: Exploration N versus Average Hypervolume for PLS-MOLA (5 Min, N = 5)

Concluding, a combination of the LS-KRCM, LS-KRBM and LS-KRPM is preferred. Although the obtained hypervolume using this combination shows no significant improvement over using LS-KRBM only, it is expected to deliver a more diverse set of neighborhood solutions. Furthermore, a KT value of 8 results in the highest hypervolume, although the difference is again not significant. Finally, the running time of the exploration N experiments was too short to produce useful results. For now, an Exploration N value of 10 has been chosen to conduct future experiments.

9.2.5 Reparation Strategy

The fifth subquestion of RQ 1 states "What is an efficient strategy for repairing non-feasible solutions?". The reparation strategy is determined by the reparation procedure and the reparation operators being used. The reparation procedure is discussed in Chapter 6.9 and repeatedly applies reparation operators to repair an invalid solution. In this research, only operators for the allocation ranges constraint have been designed and examined.

At first, the reparation operators LR-KRCM and LR-KRBM were compared in run 42 to 44 (Table G.8). The results of these runs are shown in Figure 9.6 and show the resulting average hypervolumes when using either one or both of these operators for 5 minutes on the Brazil case. Using the LR-KRBM operator results in a slightly higher hypervolume (3.698×10^{14}) than using only the LR-KRCM operator (3.650×10^{14}) or both (3.696×10^{14}). When performing a two-tailed independent sample t-test on the hypervolumes obtained from using only LR-KRBM or LR-KRCM, this results in a p-value of below 0.0001, showing a significant difference. However, the hypervolumes from using only LR-KRBM or both LR-KRBM and LR-KRCM result in a p-value of 0.541648, showing no significant difference. In order to choose what operators to use for the final algorithm comparison in Test Case VIII, we will therefore look at the diversity that is expected to be created in the repaired solutions. In that case, LR-KRBM and KRCM will be preferred above LR-KRBM, as these two combined are expected to (more easily) generate more

diverse solutions than solely LR-KRBM. Also, the LR-KRBM focuses mainly on the compactness objective, and could deliver lower hypervolumes when compactness is not being optimized.

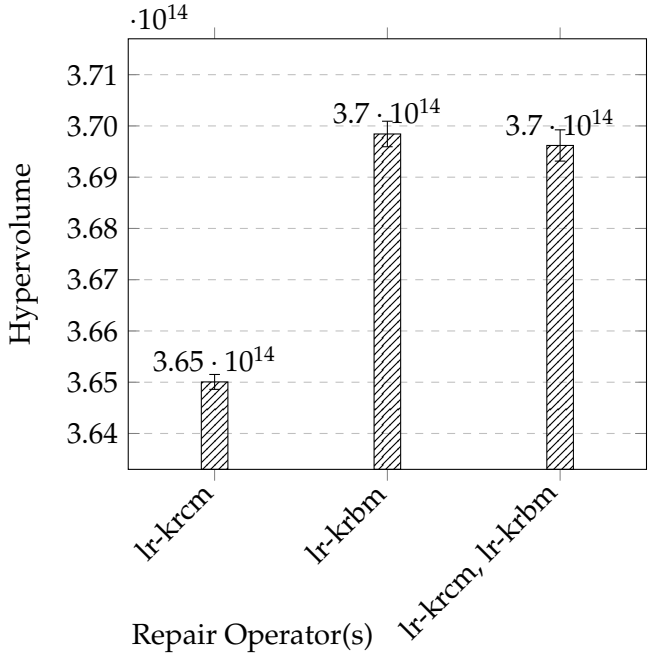


FIGURE 9.6: Average Hypervolume per Repair Operator(s) for PLS-MOLA (5 Min, N = 5)

In order to gain more insight in the complexity of running the reparation operators, run 45 to 47 were executed. The operators were used to execute 1000 iterations on the Brazil case, of which the total running times were measured. As can be seen in Table G.9 of Appendix G, this took on average 9.55×10^5 ms when using the LR-KRBM repair operator and 3.47×10^5 ms when using the LR-KRCM operator. This could be expected, as the LR-KRBM requires the operator to search for a boundary cell instead of being able to select any dynamic cell at random. However, applying the LR-KRBM operator seems to result in solutions with higher objective values, as the final hypervolume obtained is above the one of LS-KRCM (Figure 9.6). This is also expected, as modifying only boundary cells promotes the compactness.

Next up, are the results regarding the Reparation KC parameter of the repair operators. This parameter states the number of (boundary) cells that are competing in one repair operation. The repair operator will eventually apply the reparation adjustment to the cell that leads to the highest objective values. As was discussed in Chapter 8.3.5 (Test Case IV), the Reparation KC parameter was examined by repairing the initial solution and looking at the resulting solution quality and reparation time. This was done, as it was uncertain to what extent a high KC value can increase the reparation time. In case this would be extremely high, running the algorithm for a predefined number of iterations or time would bring difficulties. In Figure 9.7, the average hypervolume of the initial solution is shown after being repaired with different Reparation KC values (Table G.10). As expected, adding more cells to the competition results in a higher final hypervolume. The increase tends to be logarithmic, whereas the amount of increase in hypervolume decreases

with higher Reparation KC values. However, in order to determine a suitable Reparation KC value for PLS-MOLA, the average reparation time also has to be taken in account. These were also measured in run 48 to 53 (Table G.11), and visualised in Figure 9.8. As can be seen, the reparation time of the initial solution increases linearly with the Reparation KC value. As so, a trade-off will have to be made between the reparation time and resulting hypervolume. As can be seen in Figure 9.7, the hypervolume increases relatively fast when raising the reparation KC from 0 (3.423×10^{14}) to 100 (3.744×10^{14}). After a KC of 100, further raising the KC value becomes much less effective (to only 3.756×10^{14} at KC = 200). As the reparation time still increases linearly with the value of KC, it therefore becomes less attractive to choose higher values of KC. Therefore, a KC value of 100 is considered to be an acceptable trade-off for the eventual algorithm comparison in Test Case VIII. Note, that the 'optimal' trade-off will always differ per user, considering his/her optimization goals.

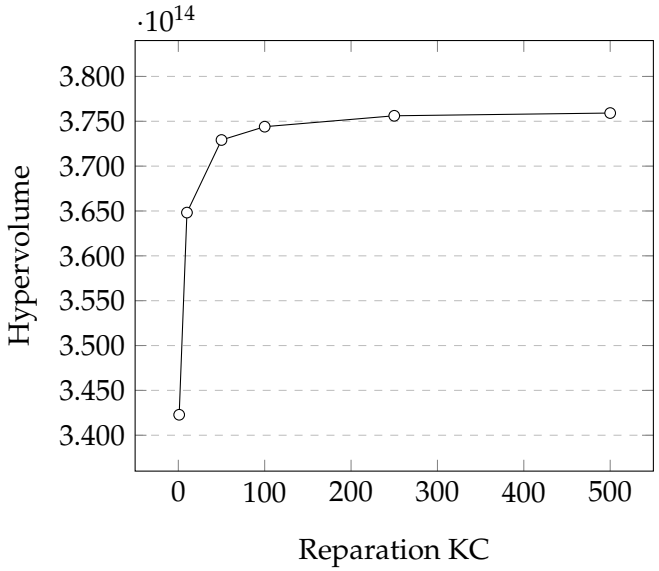


FIGURE 9.7: Reparation KC versus Average Hypervolume of Repaired Initial Solution for PLS-MOLA (N = 5)

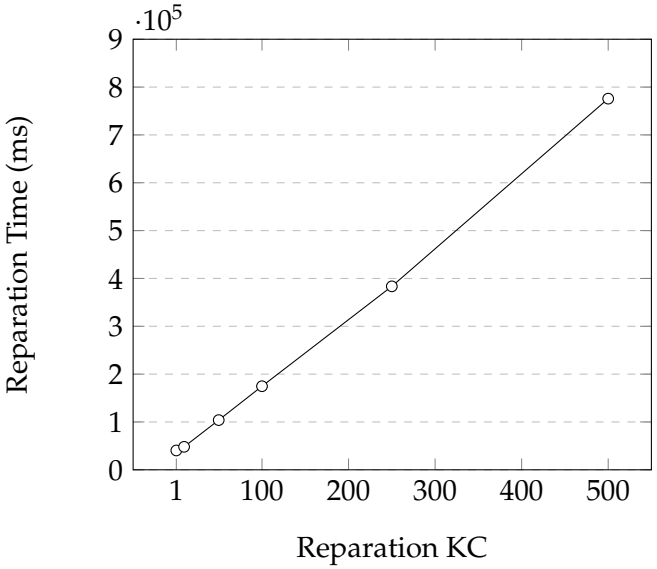


FIGURE 9.8: Reparation KC versus Average Reparation Time of the Initial Solution for PLS-MOLA (N = 5)

Besides the Reparation KC parameter, one more parameter regarding the reparation operators was being examined, called the Reparation BT parameter. If set to true, the reparation operator will not only perform a competition between different cells, but also between different types. This is only the case, when multiple types can be chosen for a cell that would aid to repairing the solution. If so, the type will be chosen that delivers the highest objective values. If the BT setting is set to false, this type will be chosen at random. In Run 54 and 55, the influence of the BT parameter on the hypervolume was examined (Table G.12) when PLS-MOLA was run on the Brazil case for 5 minutes. As shown in Figure 9.9, the PLS-MOLA algorithm reaches a slightly higher hypervolume with BT set to true (3.686×10^{14}) than with BT set to false (3.625×10^{14}). A two-tailed independent sample t-test between both settings results in a p-value of below 0.00001. Since this is below 0.05, we can state that the algorithm performs significantly better with the Reparation BT parameter set to true. This proves that the costs of performing more type checks are overruled by the benefits of obtaining repaired solutions with better objective values. As so, the Reparation BT setting set to true is considered to result in the most efficient PLS-MOLA optimization process.

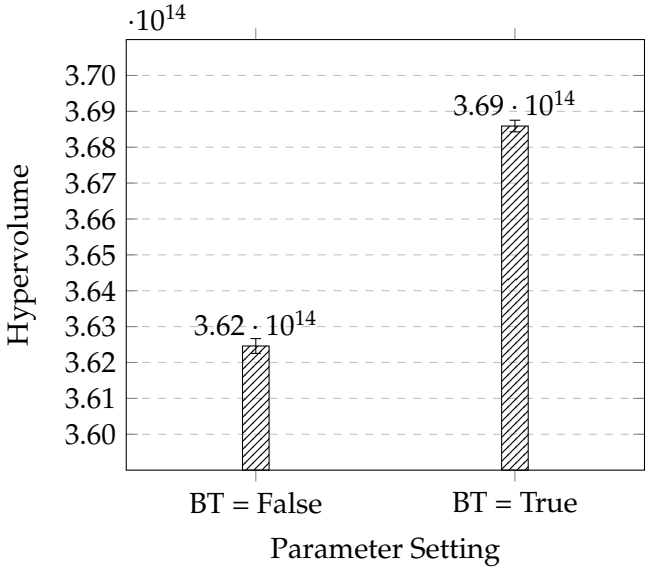


FIGURE 9.9: Average Hypervolume per BT Setting for PLS-MOLA (5 Min, N = 5)

Finally, one more reparation related setting for the PLS-MOLA algorithm has being investigated, namely the Allow Repair (AR) parameter. When set to false, no reparations are allowed throughout the optimization process (except for the initial solution). This means, that all newly found solutions that are invalid, are immediately discarded (and new ones are created until a valid one is found). Run 56 and 57 applied PLS-MOLA to Brazil for 5 minutes, either with AR set to true or false (Table G.13). The results in Figure 9.10 show the resulting average hypervolumes. As can be seen, AR set to true results in a slightly higher average hypervolume (3.688×10^{14}) than AR set to false (3.674×10^{14}). A two-tailed independent sample t-test result in a p-value of 0.002474, showing the difference is significant (below 0.05). As so, PLS-MOLA performs significantly better with AR set to true. The advantage of AR set to true, is that certain solutions can easier be 'reached' when constraint violation and repairing is allowed. Else, it might take multiple local search steps to reach the same solution. Also, the reparation operators used are designed to even further improve solutions (for instance with high Reparation KC values) when repairing. Finally, in case violations occur often, a lot of new solutions might have to be generated until a valid one is finally found. This can explain the PLS-MOLA optimization process becoming more efficient when allowing reparations.

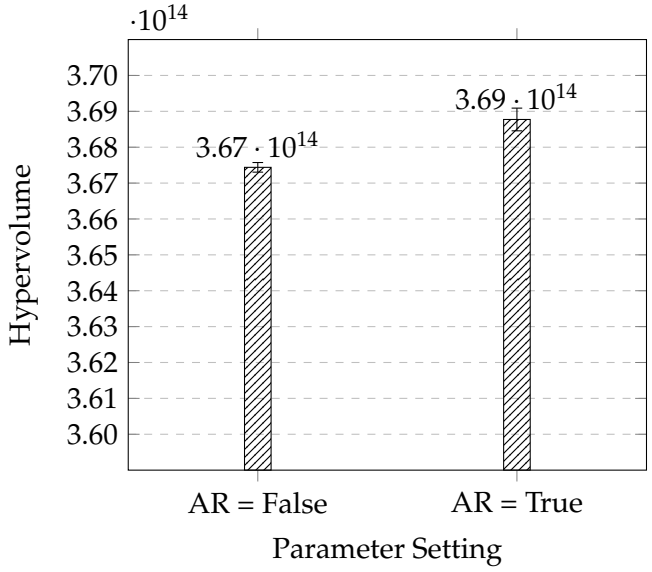


FIGURE 9.10: Average Hypervolume per AllowRepair (AR) Setting for PLS-MOLA (5 Min, N = 5)

To conclude, a combination of the LR-KRCM and LR-KRBM repair operators is preferred. Although this combination does not provide a significant improvement over using LR-KRBM only, it is expected to bring more diverse solutions (and less dependent on the compactness objective). The Reparation KC parameter brings a trade-off between reparation quality and reparation time. Although this is user dependent, a Reparation KC value of 100 is considered to bring an acceptable trade-off for this research. Furthermore, the operators are proven to performs significantly better with the Reparation BT parameter set to true. Finally, allowing reparations during the optimization process (AR = True) results in a significantly higher hypervolume.

9.2.6 Acceptance Criteria

The final subquestion of RQ 1 states "What are efficient archive acceptance criteria for non-dominated solutions?". The acceptance criteria determine whether a newly found is added to the archive or not. At all times, solutions can only be added to the archive in case they are not dominated by any other solution currently in it. In this research, two additive criteria have been examined that can be combined with this criterion. The first additive criterion is activated with the NDS parameter. If set to true, solutions can only be added in case they dominate the current solution being explored. Otherwise, solutions can also be added if they do not dominate this solution (and are not dominated by other ones either). In run 58 and 59, the PLS-MOLA algorithm was run on the Brazil case for 5 minutes (Table G.14). In Figure 9.11, the resulting hypervolumes are shown. If NDS is set to true, a slightly higher hypervolume is obtained (3.719×10^{14}) than with NDS set to false (3.688×10^{14}). A two-tailed independent sample t-tests results in a p-value of below 0.00001. As this is below 0.05, the NDS criterion appears to bring a significant improvement in the resulting hypervolume. Due to the more strict NDS criterion, less solutions are accepted to the archive, but the average quality will be higher. This increase in hypervolume will happen in each iteration over again, and eventually add up to each other as one solution

of the previously more quality archive is the start of the next one.

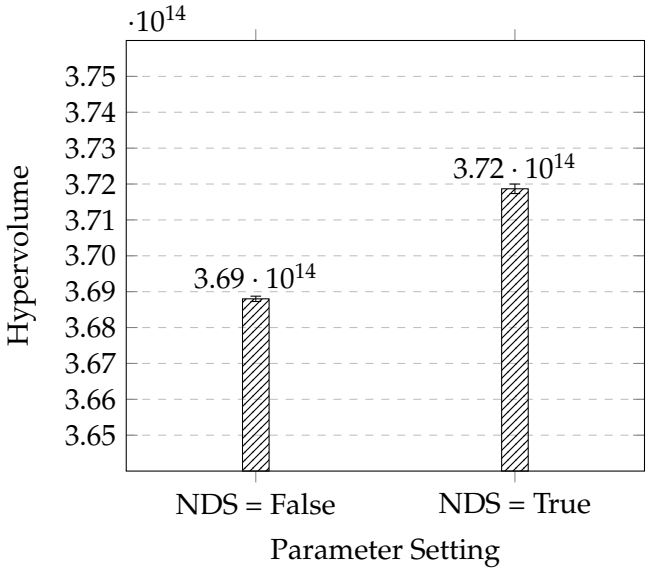


FIGURE 9.11: Average Hypervolume per NDS Setting for PLS-MOLA (5 Min, N = 5)

However, as was already mentioned in Test Case V (Chapter 8.3.6), the NDS criterion also brings a downside; it will result in smaller archive sizes. Especially in later stages, when improvements become harder to find, this might result in the algorithm not being able to find any improvements (before the termination criterion are met) faster. The average number of solutions in the final archive is shown in Figure 9.12. With NDS set to false, the final archive has 17.8 solutions on average. With NDS set to true, the average number of solutions in the archive decreases to only 4 (Table G.15). A two-tailed independent sample t-test results in a p-value of 0.0193, which is below 0.05 and so proves a significant difference in the archive size. Although the average hypervolume increases, the chance of not finding any improvements in an iteration also raises. Not finding any improvements could terminate the PLS-MOLA algorithm, which could have been prevented by also accepting and exploring these 'lesser' solutions. Also, the diversity of the solutions decreases, as a solution now always has to increase in all objective values. This means, that you will less easily obtain solutions that excel at only one objective. It is up to the user to decide, whether this is desired or not. In this research, the NDS will preferably be set to false; the slight increase in hypervolume (1.01%) is not considered to outweigh the enormous decrease in archive size (345.00%).

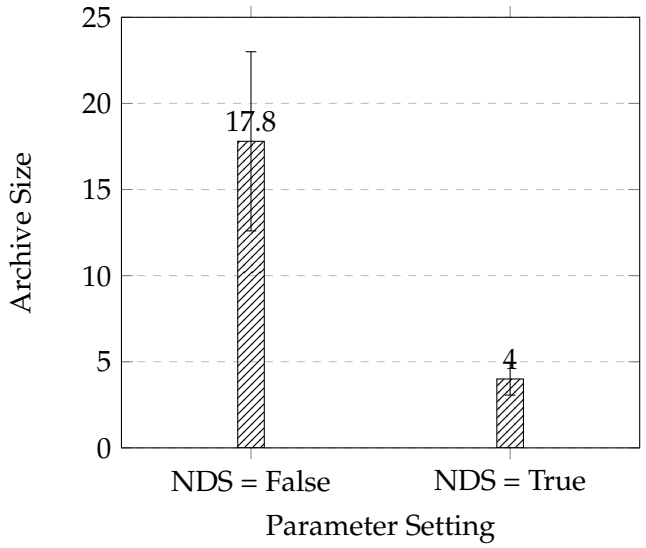


FIGURE 9.12: Average Number of Solutions per NDS Setting for PLS-MOLA (5 Min, N = 5)

Finally, one more acceptance criterion was examined; the maximum archive size. In case the maximum archive size is violated, the solution with the lowest crowding distance will be removed. In run 60 to 66, the PLS-MOLA algorithm was run on Brazil for 5 minutes using different maximum archive sizes (Table G.16). The resulting hypervolumes are shown in Figure 9.13. As can be seen, the obtained hypervolumes are close to each other, and seem to slightly increase as the maximum archive size raises. A maximum archive size of 5 delivered an average hypervolume of 3.687×10^{14}, whereas a maximum archive size of 100 gave an average hypervolume of 3.693×10^{14}. A two-tailed independent sample t-test upon these hypervolumes results in a p-value of 0.077072, showing no significant difference (above 0.05). Limiting the number of solutions, similar to the NDS case, might also result in the PLS-MOLA algorithm terminating more easily. Accepting a bigger and more diverse archive will decrease this risk. Therefore, as the size does not seem to affect the resulting hypervolume, the archive will not be limited in the final comparison of Test Case VIII.

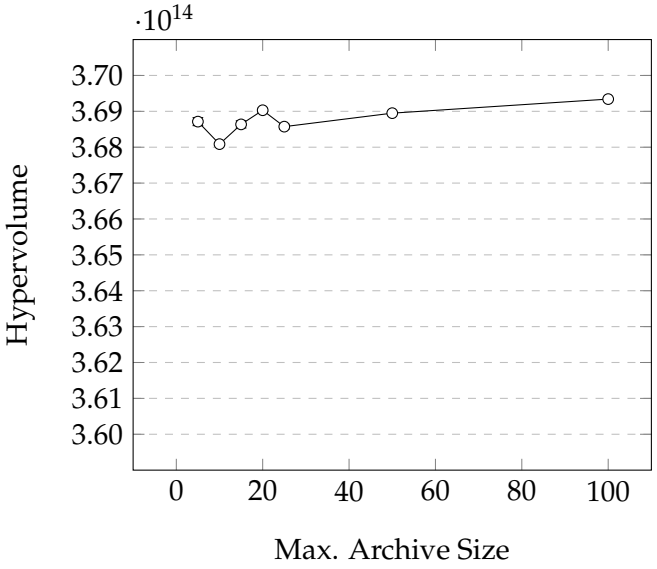


FIGURE 9.13: Maximum Archive Size versus Average Hypervolume for PLS-MOLA (5 Min, N = 5)

In summary, the NDS criterion delivers a small increase in average hypervolume (1.0%), but an enormous decrease in the average archive size (345.0%). As this could result in the algorithm terminating more easily in later stages, the NDS criterion is not preferred in the remaining of this research. Finally, the maximum archive size does not seem to influence the resulting hypervolume. An infinite maximum archive size will therefore be preferred, to not limit the archive size and again decrease the chance of the algorithm terminating early.

9.3 RQ II: PLS Objective Scalability

The second research question states "How does the efficiency of PLS for MOLA scale with the number of objectives?". In theory, an infinite number of objectives can be optimized by PLS simultaneously. However, the more objectives are being optimized, the less efficient the objectives will be optimized individually. The local search operators will have to divide their attention over more objectives, and solutions will be added to the archive more quickly as they are dominated less often. In the previous experiments, the PLS-MOLA algorithm was run on the Brazil case with two objectives; potential yield and compactness maximization. In this section, the effects of the addition of a third objective, namely carbon stock maximization, are discussed.

At first, the relation between the number of objectives and the average archive size was examined. This was done in run 67 and 68, in which the PLS-MOLA algorithm was run for 1000 iterations with either two or three objectives (Table G.18). The resulting average archive sizes are shown in Figure 9.14. When only compactness and potential yield are being maximized, an average archive size of 15.0 solutions was obtained. Now, when adding the carbon stock maximization objective, the average archive size increased to over 1135.4 solutions. A two-tailed independent sample t-test results in a p-value of below 0.00001. As this is below 0.05, the increase is significant. An increase in archive has both advantages and disadvantage. Disadvantage, is the increase in memory required to run the algorithm. All solutions

have to be stored simultaneously. Especially in later stages, this could result in high memory demands (as each solutions stores the number of adjustments made). Also, archive related operations, such as checking whether a new solution is dominated, will require more operations. In order to counter these negative effects, a maximum archive size could be used. On the other hand, a large archive size decreases the chance of the algorithm terminating early. The fast expanding archive can lead to more diverse solutions, exploring larger sections of the solution space.

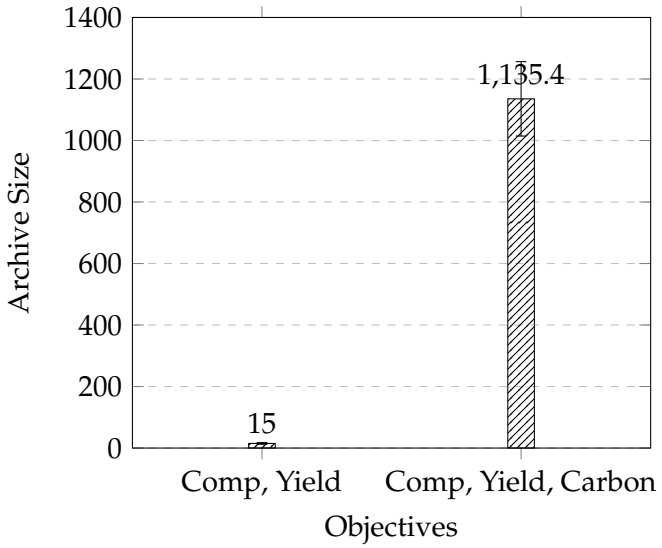


FIGURE 9.14: Average Archive Size per Objective(s) Optimized for PLS-MOLA (1000 Iterations, N = 5)

More objectives result in less attention of the local search operators for each objective individually. As the current local search operators randomly pick an objective to focus on (Chapter 6.8), the chance of being picked decreases from 50% to only 33%. In run 69 and 70, the hypervolume of only compactness and potential yield have been measured when PLS-MOLA was run on Brazil for 5 minutes with either 2 or 3 objectives (Table G.19). Hereby, the carbon stock values of the runs with 3 objectives have been neglected. As shown in Figure 9.15, the addition of a third objective decreases the hypervolume of compactness and potential yield from (on average) 3.691×10^{14} to 3.572×10^{14}. This can be considered as a rather large difference, as we are only 5 minutes in the optimization process. A two-tailed independent sample t-test shows results in a p-value of below 0.00001, showing the difference is significant. This confirms our expectations regarding the lower optimization quality per objective individually when more objectives are added. As so, it is always important to outweigh whether the addition of an extra objective is worth this decrease in optimization quality. Optionally, objectives can be combined to prevent the number of objectives from increasing. Another option, is to do multiple runs given different sets of objectives and/or using the outputs of previous runs as input. This will be discussed in more detail in Chapter 11.

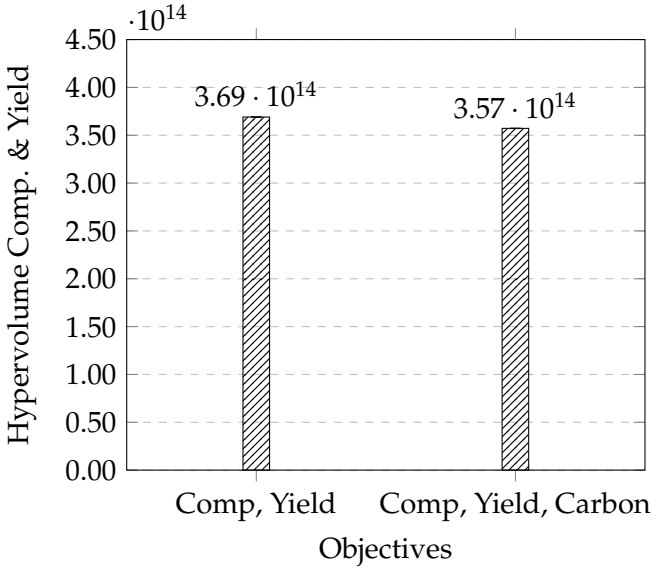


FIGURE 9.15: Average Hypervolume Comp. & Yield per Objectives Optimized for PLS-MOLA (5 Min, N = 5)

Finally, two runs were executed to gain more insight in the relation between the number of objectives and the time to complete an iteration in PLS-MOLA. As so, run 67 and 68 were executed (Table G.17), in which the algorithm was run for 1000 iterations on the Brazil case with either 2 or 3 objectives and the running times were measured. The results are shown in Figure 9.16, and show an outcome that one might not expect; the PLS-MOLA algorithm finished earlier when optimizing 3 objectives (4.26×10^5) than when optimizing only 2 objectives (8.36×10^5). A two-tailed independent sample t-test shows that the results are significant with a p-value of below 0.00001. In general, optimizing 3 instead of 2 objectives would require the algorithm to perform more operations per iteration (for example to check dominance). However, the costs of these extra operations seem to be outweighed by another aspect; the chances of violating a constraint. Now, the current decrease in running time can be explained as follows: the potential yield objective asks for much more repair operations than both compactness and carbon stock maximization. This is caused by the allocation ranges constraint, which is closely related to the potential yield objective; only boundaries are set upon the land-use types that influence the potential yield. Therefore, the need (chance) of a reparation is the highest after a local search operator changes a yield related land-use type. Now, the local search operators choose one random objective to focus on. Chances are the highest to pick yield maximization when optimizing only 2 objectives (50%) compared to when optimizing 3 objectives (33%). As so, optimizing 2 objectives results in more constraint violation, requiring more repair operations, which again results in longer running times. In order to confirm this theory, two more runs have been performed. The first one maximized the compactness and carbon stock, and resulted in an average running time of 4.30×10^5ms (N = 5). The second one maximized the potential yield and carbon stock, which resulted in an average running time of 5.02×10^5ms (N = 5). As expected, the runs optimizing two objectives among one was yield maximization ended up having longest and second-longest running time. This shows that yield maximization is the most 'costly' objective, explaining the results of Figure 9.16.

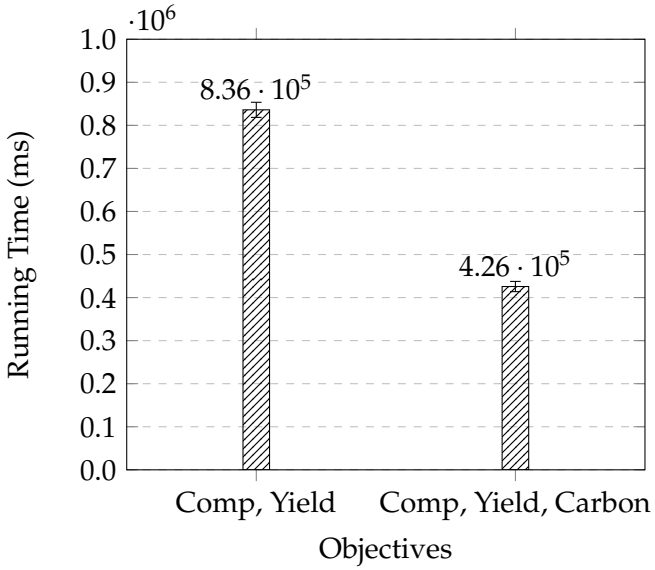


FIGURE 9.16: Average Running Time per Objective(s) Optimized for PLS-MOLA (1000 Iterations, N = 5)

To conclude, the addition of more objectives can first of all significantly increases the archive size. As so, more memory space will be required. Furthermore, the individual optimization quality per objective reduces, as the search operators have to focus on more objectives. Therefore, one should consider whether this is worth the addition of more objectives. Combining objectives or performing different runs in a row could be an alternative (Chapter 11). Finally, although optimizing more objectives requires more operations per iteration, the running times depend strongly on the 'costs' of the objectives being optimized. In case an objective requires more repair operations, the running time can rise significantly. The higher the chance of a local search operator choosing such an expensive objective to optimize, the longer the expected running times will be.

9.4 RQ III: IPLS Optimization

The third research question states "How can PLS most efficiently be processed into a global IPLS algorithm for MOLA?". As shown in Algorithm 16, the IPLS algorithm iteratively calls the PLS algorithm. As so, the answers to RQ 1 (optimizing PLS) are also part of an efficient IPLS setup; the settings used to iteratively call the PLS algorithm will be based upon these results. In Table 9.1, the settings found most efficient in each section of Chapter 9.2 are summed up. Note, that these parameters were optimized using the large Brazil case. As so, these might not be optimal for the smaller Centre West and Sul Goiano case. However, as the smaller cases are assumed to be a relatively good representation of Brazil (Chapter 8.2.5), and optimizing PLS for each case separately is out of the scope of this research, the parameter settings will be reused.

TABLE 9.1: Selected PLS-MOLA Parameters Settings

Id	Parameter	Value
6	Selection Operators	s-cd
7	Search Operators	ls-krcm, ls-krbm, ls-krpm
8	Repair Operators	lr-krcm, lr-krbm
9	Search KT	8
10	Repair KC	100
11	Repair BT	True
12	Allow Repair	True
13	NDS	False
14	Max. Archive Size	∞
15	Exploration N	10

As discussed in Test Case VII (Chapter 8.3.8), we will first determine what cases IPLS is eligible for. Runs longer than 2.5 hours are considered to be out of the scope of this research. Therefore, IPLS is only eligible for a case when the PLS algorithm is able to find a local optimum at least once in 2.5 hours (on average). To find out for which cases this holds, the PLS algorithm was run on the Brazil case and the two subcases (5 times each) using the settings of 9.1. Only in case the PLS algorithm terminated within the timespan of 2.5 hours, meaning a local optimum was found (approximated), the problem case will be eligible for IPLS. At first, the Brazil case was examined, which is the largest case study with 342815 dynamic cells. The PLS-MOLA algorithm was run on the Brazil case 5 times and the archive was stored each 15 minutes. The resulting hypervolumes are shown in Table H.1 and visualised in Figure 9.17. As can be seen, the PLS-MOLA algorithm did not terminate within the timespan of 2.5 hours when applied to the Brazil case. As so, IPLS will not be run on Brazil in this research. The reason PLS-MOLA was not able to find a local optimum in this timespan, is due to the size of the case. The Brazil case is (significantly) larger than the cases being optimized in previous research regarding MOLA (Chapter 4). For example, the two most common MOLA problems discussed in Chapter 3.6, only existed out of 400 and 3150 dynamic cells. The solution space of the Brazil case is much larger, and so it takes much more operations to reach an optimum. Since IPLS-MOLA will not be run on the Brazil case, the performance of the NSGA-II algorithm will be compared to PLS-MOLA (Chapter 9.5).

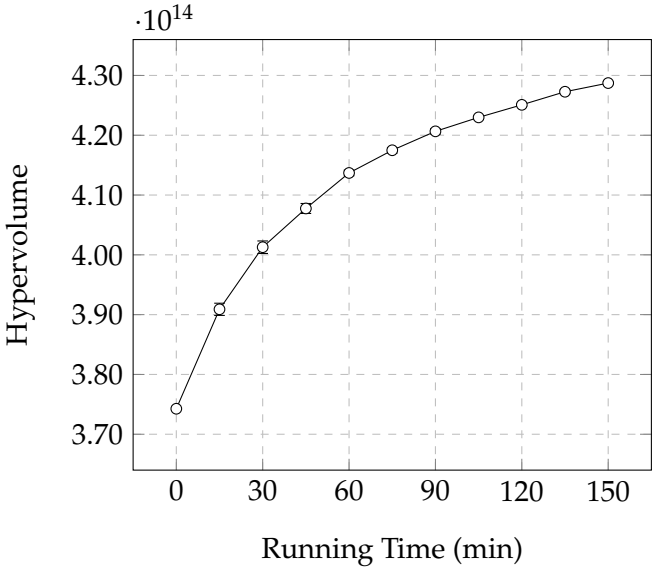


FIGURE 9.17: Average Hypervolume versus Running Time for PLS-MOLA for Brazil (n = 5)

As can be seen in Figure 9.17, it is difficult to estimate at what time a local optimum would have been found. This will approximately be around the time that the increase in hypervolume stagnates. However, the hypervolume still seems to increase steadily at the end. In order to successfully run the IPLS algorithm, longer running times will be necessary. Alternatively, more powerful hardware could be used to speed up the PLS convergence. Currently, the algorithm was run on a personal computer, which is not optimized for these kinds of optimization jobs. This will be discussed later in Chapter 11. Finally, it has to be noted that the first PLS run might take longer than succeeding ones (depending on the perturbation size), as the initial solution can be further away from an optimum than a perturbed solution. This can also be seen with the Sul Goiano problem case later on in this chapter. As later iterations might take less time, IPLS might be able to perform more iterations than expected, based upon the first one only. Still, given the resources of this project, the IPLS algorithm will not be run on the Brazil case.

One of the solutions of the final archive of the PLS-MOLA algorithm is shown in Figure 9.18. The solution shows, among other developments, an increase in the amount of crops land-use in the South. As can be seen in the yield map of Figure 8.3, this is not a surprise; the potential yield of this type is the highest at this location. Statistic regarding the final archive can be found in Table H.2. The original map of Brazil had a compactness value of 6.757×10^5 and a potential yield value of 4.413×10^8. The final archive of PLS-MOLA had an average compactness value of 6.970×10^5 (+3.2%) and average potential yield value of 6.174×10^8 (+39.9%). Note that these are only averages, and the maximum objective values obtained are even higher.

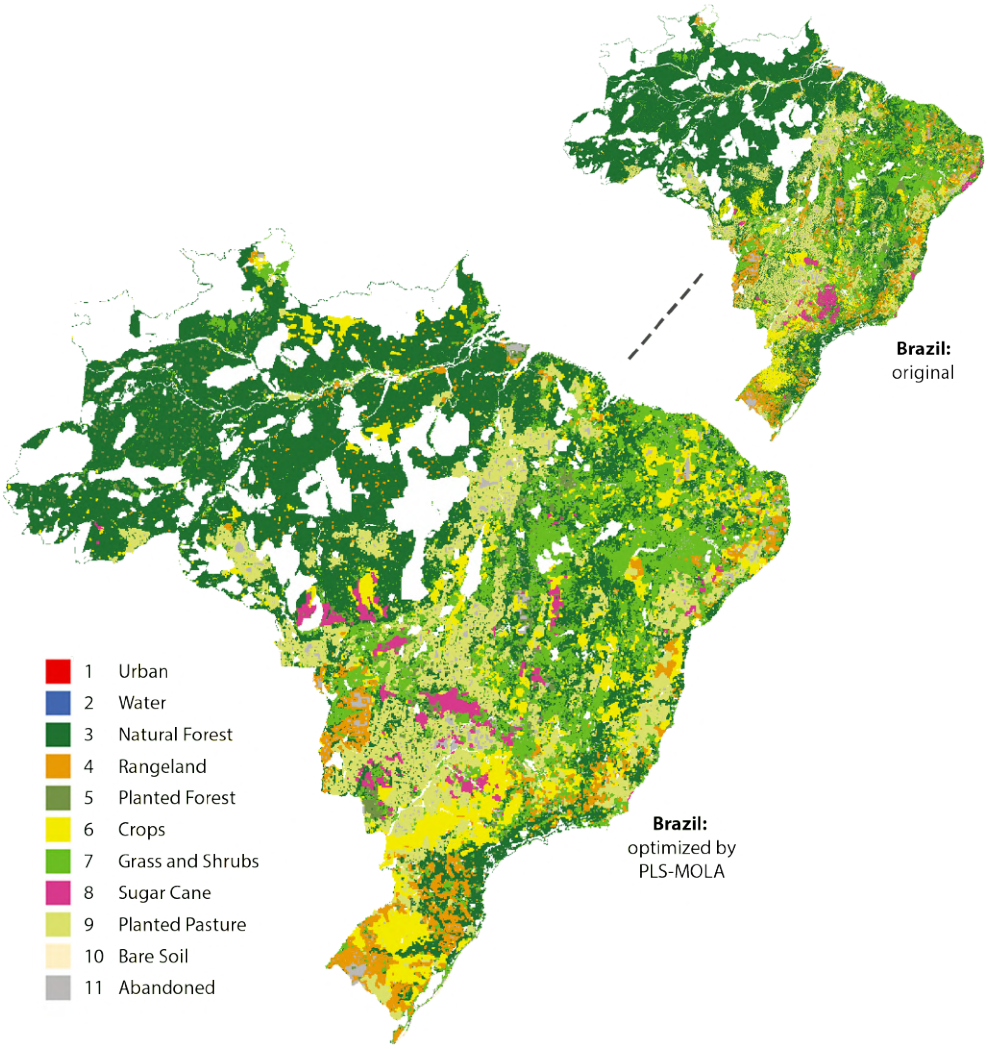


FIGURE 9.18: Brazil with PLS-MOLA

The second problem case being examined, is the medium sized Centre West case with 63827 dynamic cells. Again, the PLS-MOLA algorithm was first run 5 times for 2.5 hours with the parameters shown in Table 9.1. The resulting hypervolumes are shown in Table H.3 and visualised in Figure 9.19. Similar to the Brazil case, the algorithm did not terminate in 2.5 hours. The Centre West case is therefore also not eligible for the IPLS algorithm, given the resources of this project. Similar to the Brazil case, the Centre West case is still (much) larger than the problem cases examined in previous research.

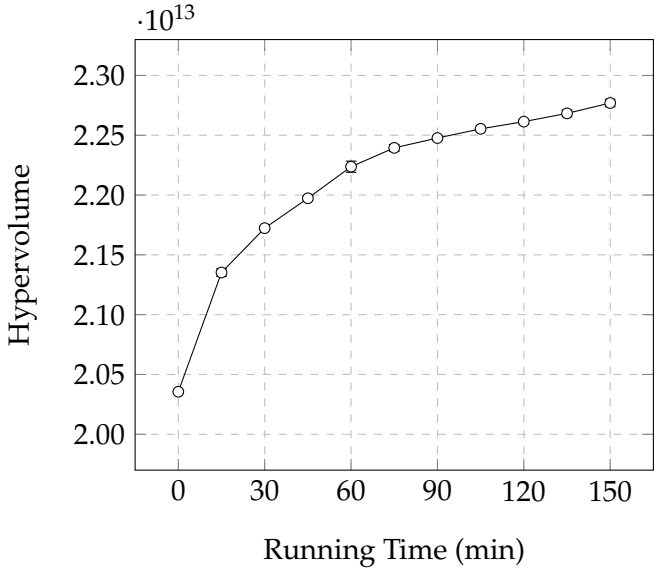


FIGURE 9.19: Average Hypervolume versus Running Time for PLS-MOLA for Centre West (N = 5)

A solution from the final archive of the PLS-MOLA algorithm when run on the Centre West case is shown in Figure 9.20. The solution shows, among other developments, a decrease in rangeland in the West and an increase in sugar cane in the South. Again, this benefits the potential yield and is therefore a logical expansion. Statistics regarding the final archive can be found in Table H.4. The original map of Centre West had a compactness value of 1.161×10^5 and a potential yield value of 1.205×10^5. The average compactness value of the final archive was 1.252×10^5 (+3.9%), whereas the average potential yield value was 1.830×10^5 (+46.2%).

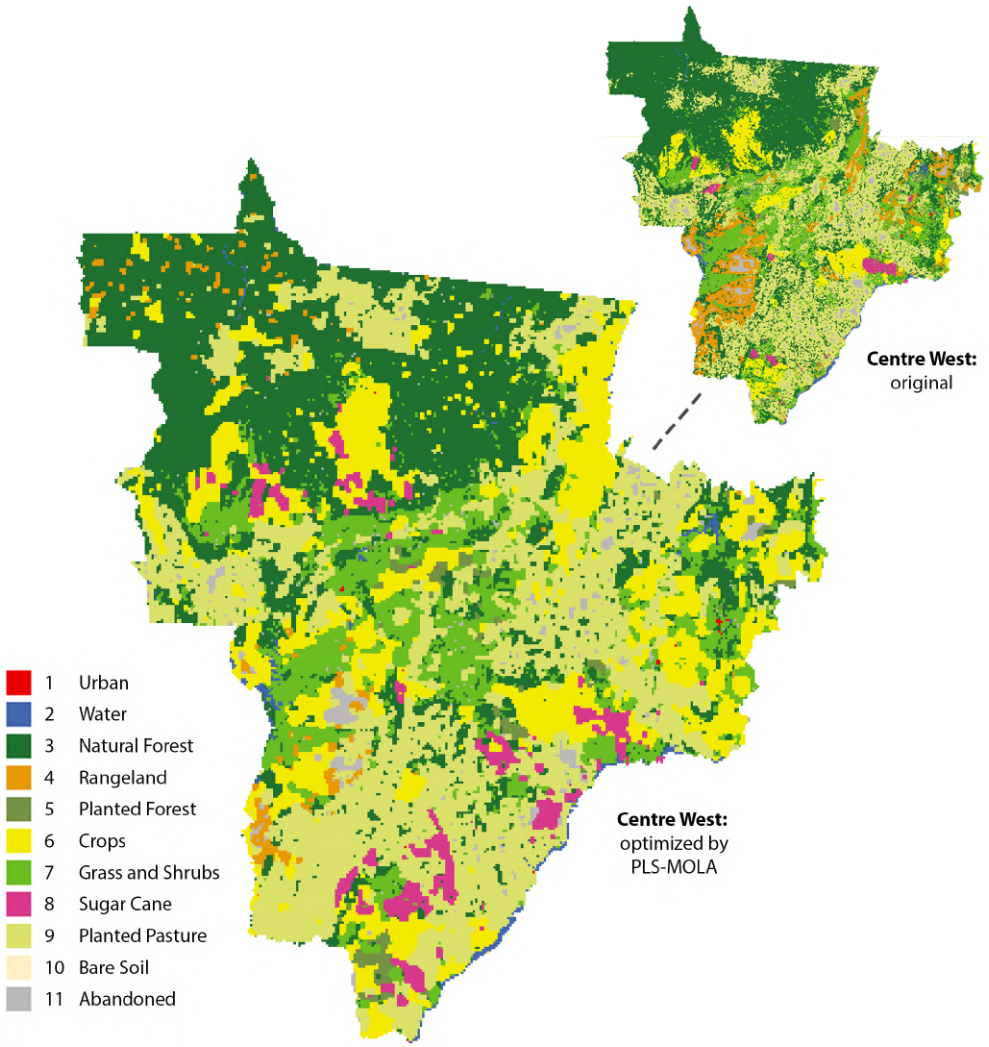


FIGURE 9.20: Centre West with PLS-MOLA

Finally, the PLS-MOLA algorithm was run on Sul Goiano, which is the smallest problem case with 5229 dynamic cells. In contrast to the Brazil and Centre West case, the PLS-MOLA algorithm did actually terminate before the maximum running time of 2.5 hours. The resulting hypervolumes are shown in Table H.5 and are visualised in Figure 9.21. Three out of five runs terminated in 45 minutes, whereas two terminated in 60 minutes. This makes an average running time of the (first) iteration of 51 minutes and makes the case eligible for IPLS. Note, that the Sol Goiano problem case is still larger than the largest common example case in previous research, discussed in Chapter 3.6. Therefore, IPLS is most likely eligible for these cases as well.

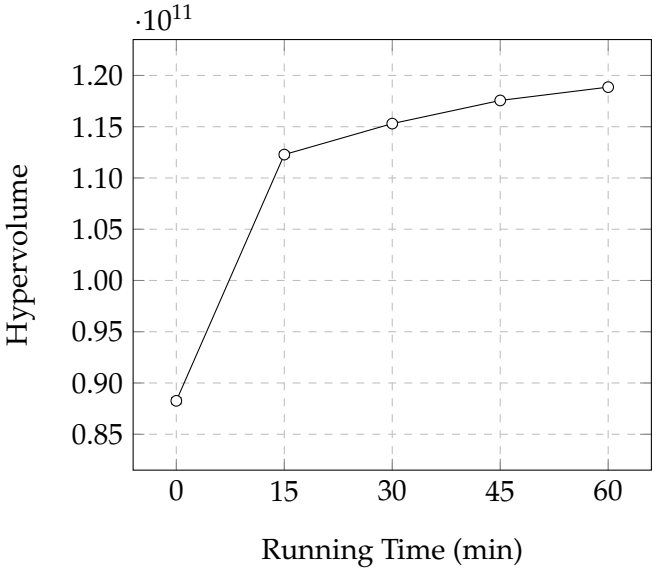


FIGURE 9.21: Average Hypervolume versus Running Time for PLS-MOLA for Sol Goiano (N = 5)

One of the solutions from the final archive of the PLS-MOLA algorithm when ran on the Sul Goiano case, is shown in Figure 9.22. As can be seen, the solution is relatively compact due to its large clusters. The amount of crops seems to have increased in the East, whereas large sections of sugar cane started to appear in the South. The original map of Sul Goiano had a compactness value of 6.751×10^3 and a potential yield value of 0.795×10^7. The average compactness value of the final PLS-MOLA archive was 7.800×10^3 (+15.4%), with an average potential yield of 1.446×10^7 (+81.9%).

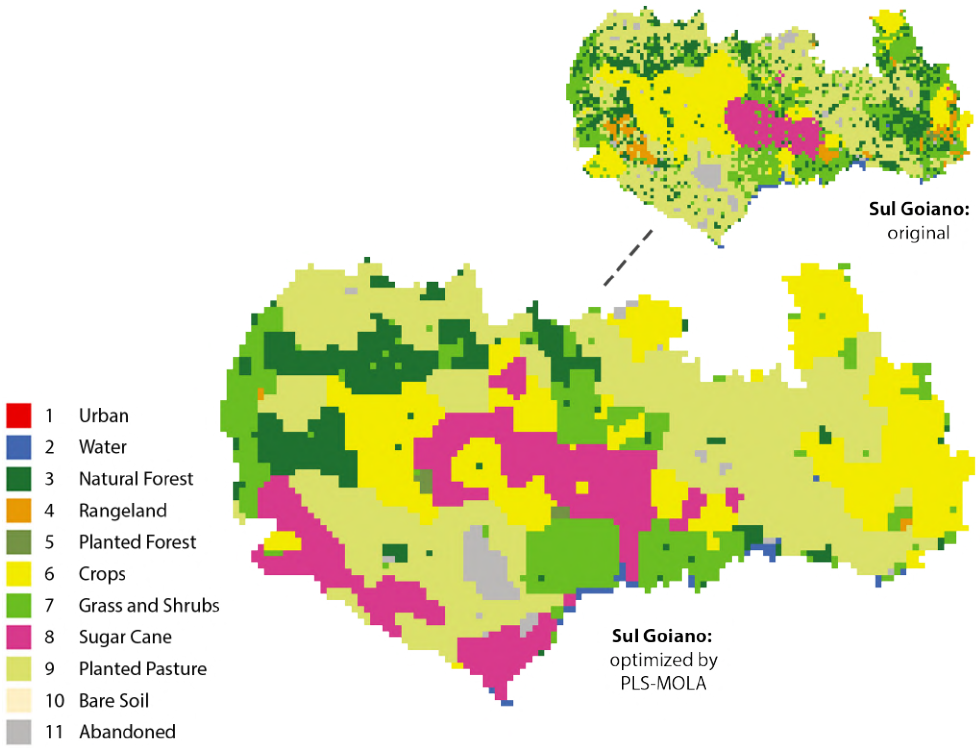


FIGURE 9.22: Sul Goiano with PLS-MOLA

As the PLS-MOLA algorithm is able to find a local optimum within the timespan of 2.5 hours, the Sul Goiano case is eligible for the IPLS-MOLA algorithm. However, before being able to efficiently run the IPLS-MOLA algorithm on the Sul Goiano case, an effective perturbation size has to be found. As discussed before, the perturbation size should be large enough to escape the local optimum of the solution, but small enough to preserve valuable solution characteristics. Therefore, run 71 to 73 have been executed, in which the hypervolumes resulting from IPLS-MOLA with different perturbation sizes were measured (Table H.7, H.8 and H.9). The results are visualised in Figure 9.23. Note, that a perturbation size of 0 or 100 are never desired; the first would not perturb the solution at all, whereas the second would create a completely random solution. As can be seen, a perturbation size of 25% results in the highest average hypervolume ($1.22907 \times 10^1 1$). The higher the perturbation size, the lower the average hypervolume becomes. A two tailed independent sample t-test between the hypervolumes obtained with a perturbation size of 25% and 50% results in a p-value of 0.034204, whereas a test between the volumes obtained with 25% and 75% results in a p-value of 0.034337. As both are below 0.05, the perturbation size of 25% proves to be significantly better in terms of solution quality. Apparently, this size allows the IPLS algorithm to best escape a local optimum, without losing all valuable information (as would be the case with for example 100%). As we will discuss next (Figure 9.24), we can indeed see that 25% was indeed enough to successfully escape a local optimum. In the future, it could be interesting to also examine perturbation sizes of below 25%.

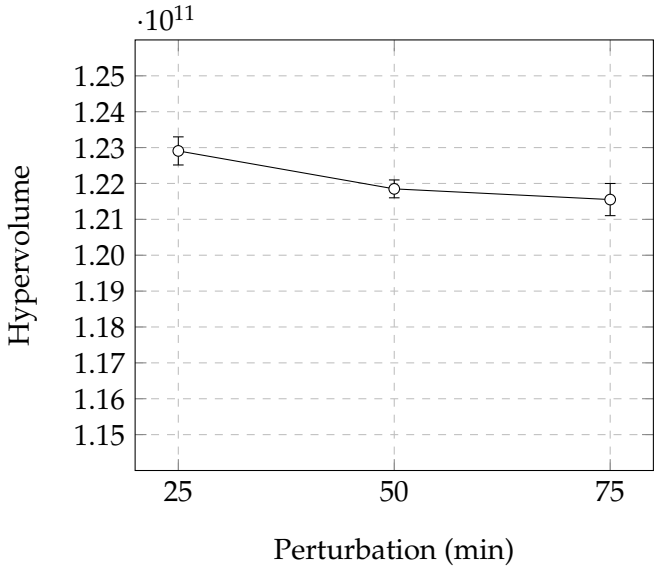


FIGURE 9.23: Average Hypervolume versus Perturbation Size for IPLS-MOLA for Sul Goiano (N = 5)

The final archive resulting from an IPLS-MOLA run with a perturbation size of 25% is shown in Figure 9.24. At the right bottom, the archive of the first PLS-MOLA run (found after 45 minutes) can be seen. By repeatedly perturbing solutions and re-applying PLS-MOLA, IPLS-MOLA was able to escape this local optimum. This eventually resulted in the IPLS-MOLA archive containing solutions that approximate one or more higher quality local optima. As can be seen, these solutions especially improve on the potential yield values. The (maximum) compactness found in later PLS-MOLA iterations is

still relatively close to the compactness found in the first PLS-MOLA execution and in some cases even lower. This shows that IPLS-MOLA is especially efficient at further improving the potential yield. Possibly, a perturbation size of 25% is still relatively large, disabling the follow-up PLS-MOLA runs to improve upon the compactness of the first run. Examining perturbation sizes of below 25% would therefore again be an interesting future research direction.

Also interesting is the number of PLS-MOLA iterations executed by IPLS-MOLA in the 2.5 hours running time. On average, the PLS-MOLA algorithm was called 7.5 times per IPLS-MOLA run with a perturbation size of 25%. The first iteration, given the initial map of the problem case, hereby takes the longest (Table H.5). All remaining iterations, given a perturbed solution from the current archive, took significantly less time; approximately 15 minutes each. This is caused due to the fact that a (25%) perturbed solution is closer to a (new) local optimum than the original map. The larger the perturbation size, the longer PLS will take to reach another local optimum.

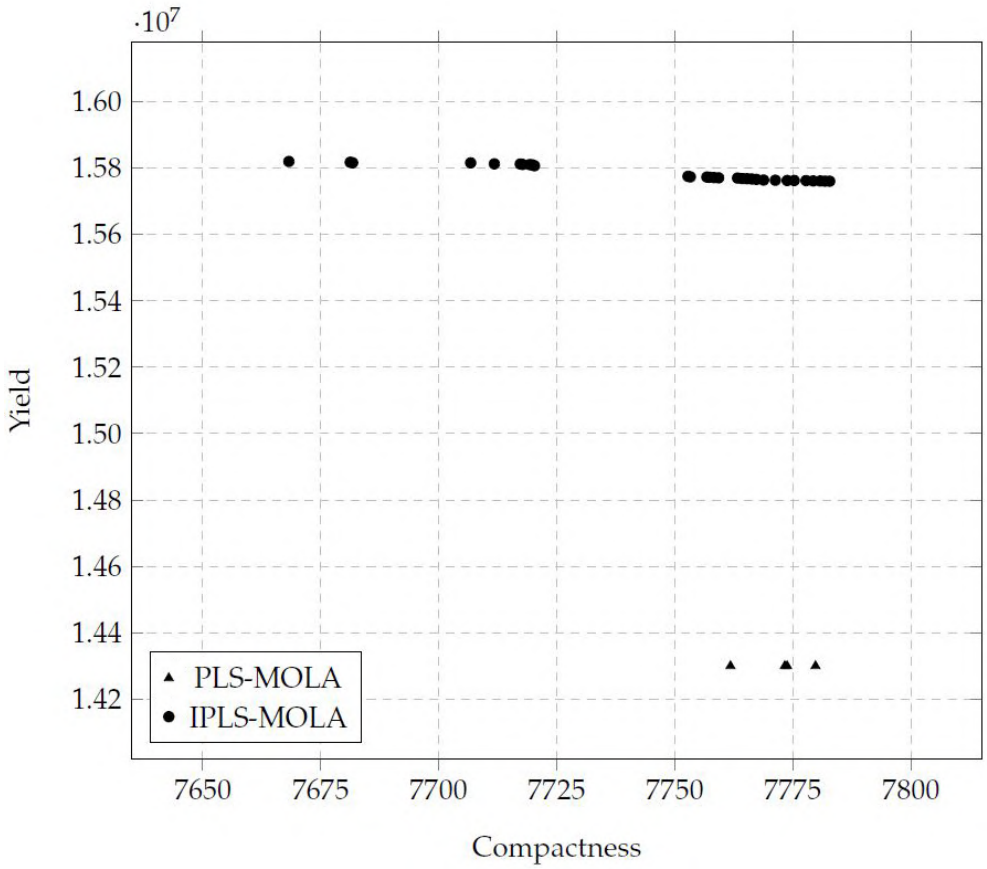


FIGURE 9.24: Final Archive of PLS-MOLA and IPLS-MOLA for Sul Goiano

One of the solutions of the final archive of IPLS-MOLA for Sul Goiano is visualised in Figure 9.25. As can be seen, the solution is more 'scattered' than the solution resulting from PLS-MOLA shown in Figure 9.22. This is the result of repeatedly perturbing solutions, and re-optimizing these again using PLS-MOLA. Over the iterations, the solutions will start to deviate more from the initial situation, as is expected with a global search algorithm. The resulting solutions had an average compactness of 7.7380×10^3 (+14.6%) and

an average potential yield value of 1.5686×10^7 (+97.3%) (Table H.10). Note again, that these are only averages; the maximum compactness found was 7.7885×10^3 (+15.4%), with a maximum yield value of 1.5880×10^7 (+99.7%).

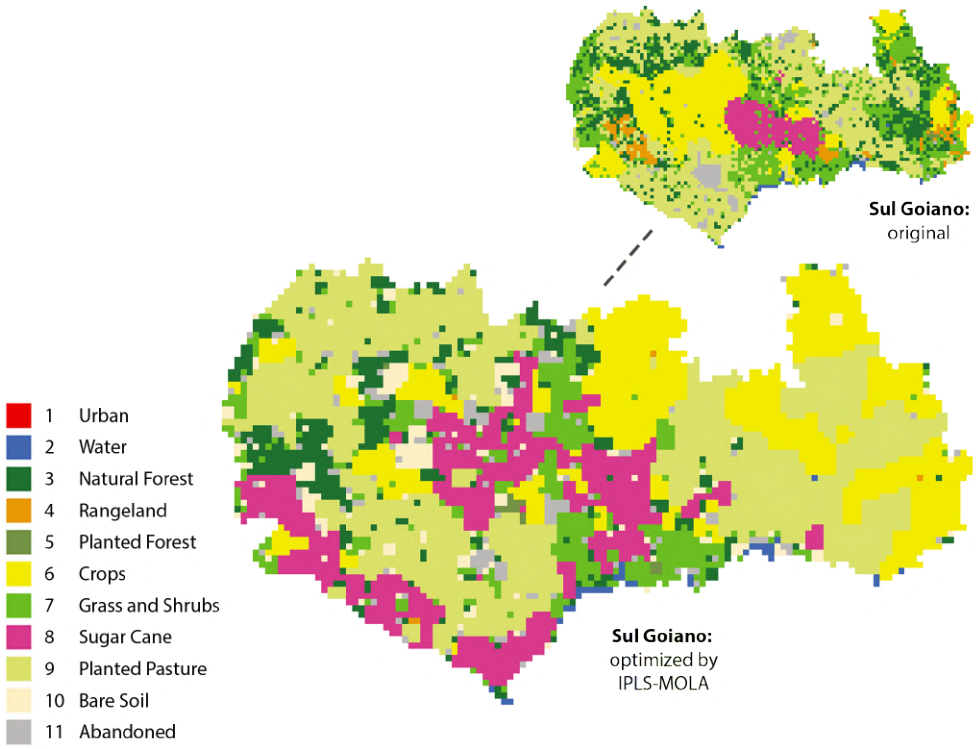


FIGURE 9.25: Sul Goiano with IPLS-MOLA

9.5 RQ IV: NSGA-II Comparison

For the final research question, the performance of IPLS-MOLA and/or PLS-MOLA is compared to the NSGA-II algorithm. The NSGA-II algorithm is currently the most applied technique for MOLA and so this will give an indication of the potential of (I)PLS-MOLA in the current MOLA domain. In case the case study was found not eligible for IPLS-MOLA (due to a long convergence time of PLS-MOLA), the NSGA-II algorithm will be compared to PLS-MOLA. The implementation of NSGA-II for MOLA was discussed in Chapter 7.3.3, and is referred to as NSGA-II-MOLA.

Similar to PLS-MOLA, the NSGA-II-MOLA was build in a modular way, which enables the user to easily apply different procedures and/or operators. As so, the search and repair operators found effective for PLS-MOLA can be used in the NSGA-II-MOLA algorithm as well. The standard parameters chosen to run NSGA-II-MOLA were discussed in Chapter 8.3.9 and are shown in Table 8.29. Some of these parameters were left open, as these will be based upon results from Test Case III and IV. The final settings for these parameters are shown in Table 9.2. Hereby, the assumption is made that the results regarding search and reparation operators for PLS-MOLA, are also usable for NSGA-II. As will be discussed in Chapter 11, it is advised to optimize these aspects solely for NSGA-II as well. First of all, the LS-KRCM, LS-KRBM and LS-KRPM operators were all selected as mutation operators,

as these also provided the best local search results for PLS-MOLA (Chapter 9.2.4). The same holds for the usage of both repair operators LR-KRCM and LR-KRBM (Chapter 9.2.5). The Mutation KT is set to 8, as this delivered the best results according to Chapter 9.2.4. Finally, the Repair KC was set to 1 and the Repair BT to true. Whereas the Repair BT set to true was proven to be most effective, the Repair KC turned out to be a trade-off between reparation speed and repaired solution quality (Chapter 9.2.5). As the NSGA-II-MOLA algorithm incorporates the disruptive C-XTD operator, it is expected to create offsprings that violate the allocation range boundaries significantly. As so, the NSGA-II-MOLA algorithm is expected to require a high number of repair operations per generation, and so the choice has been made to set the Repair KC to 1 to speed up this process. Future research could examine the influence of rising the Repair KC parameter on the convergence speed and resulting population quality of NSGA-II-MOLA.

TABLE 9.2: NSGA-II-MOLA Parameters Settings

Id	Parameter	Value
24	Mutation Operators	ls-krcm, ls-krbm, ls-krpm
25	Repair Operators	lr-krcm, lr-krbm
30	Mutation KT	8
31	Repair KC	1
32	Repair BT	True

Given the parameters of Table 8.29 and Table 9.2, the NSGA-II-MOLA algorithm was first applied to the Brazil problem case. The obtained hypervolumes are shown in Table H.11 and statistics regarding the final population are shown in Table H.12. In order to visualize the convergence of the NSGA-II-MOLA algorithm, five populations (obtained every +-30 minutes), are shown in Figure 9.26. The populations are colored darker over time; meaning the initial population is the lightest, and the final population is the darkest. Furthermore, the PLS-MOLA algorithm is added (as the Brazil case was not eligible for IPLS-MOLA), with the archives colored in the same manner. Finally, the initial Brazil map (before reparation) is shown at the right bottom as a square. First of all, let's look at the initial population of NSGA-II-MOLA (the lightest circles). Half of the initial population was created at random and ended up with yield values of approximately 5.07×10^8 and compactness values of around 5.18×10^5. The other half was created by mutating (50% of) the initial population, and ended up with similar yield values but higher compactness values of around 5.75×10^5. Reason for this, is that mutating parts of the initial clusters still results in larger clusters than when each cell is given a random land-use type. However, both types of initial solutions still end up with lower a compactness than the initial solution, as no compactness persevering initialization operators were used. The follow-up populations especially show improvements in the potential yield values of the solutions. These rise up to about 5.38×10^5. It is expected that this is caused by the C-XTD crossover in combination with the reparation operators; these allow for a re-arrangement of large sections of Brazil, with the possibility to allocate the new cells in favor of the yield production. On the other hand, the crossover operators tends to be less effective considering the compactness objective. It is suspected that this is also caused by the disruptive C-XTD operator. Re-arranging and repairing large sections can easily lead to the shattering of clusters, decreasing

the compactness. A possibility to counter this issue, is the addition of more local search operations to promote the compactness of the disrupted areas. Eventually, only a few solutions improve on the compactness of the initial population, whereas all solutions have a much lower compactness value than the initial map of Brazil. Overall, the average yield potential of the final population is 5.173×10^8 with an average compactness of 5.700×10^5. This is an average increase in yield compared to the initial solution of 17.2%, with an average decrease in compactness of 15.6%.

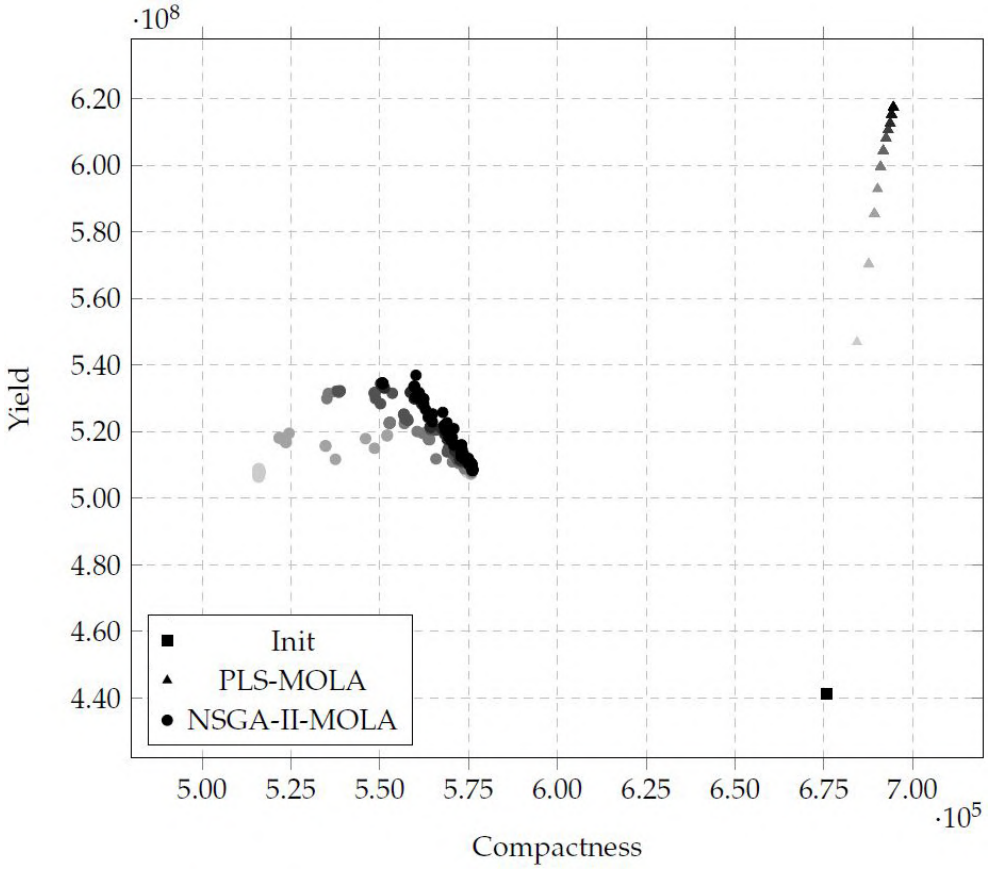


FIGURE 9.26: Convergence of PLS-MOLA and NSGA-II-MOLA for Brazil

Now, lets compare the performance of the NSGA-II-MOLA algorithm to the performance of the PLS-MOLA algorithm. The solutions in the final archive of PLS-MOLA had an average compactness value of 6.970×10^5 and an average potential yield value of 6.174×10^8. Both are above the average objective values of the solutions resulting from NSGA-II-MOLA. In fact, all of the solutions in the final archive of PLS-MOLA dominate the solutions in the final population of NSGA-II-MOLA. As so, we can state that given the 2.5 hours of running time, the PLS-MOLA algorithm performs best in terms of solution quality. Also, if we look at the convergence of both algorithm, it does not look like the NSGA-II-MOLA algorithm will outperform PLS-MOLA with longer running times either; the differences in objective values are still relatively large compared to the improvements found in the latest generations. Since the NSGA-II-MOLA algorithm did and most like will not improve upon the local optimum that is being approximated by the PLS-MOLA algorithm, the algorithm does not seem to be able to effectively perform a 'global' search.

It is difficult to state why the NSGA-II-MOLA fails to explore the solution space effectively; possibly due to inefficient crossover operators, which are not suitable for the current problem type and/or size. The only advantage NSGA-II-MOLA offers, is the fact that the final populations exists out of a larger and more diverse set of solutions compared to the relatively small PLS-MOLA archive.

Finally, one of the solutions of the NSGA-II-MOLA algorithm is shown in Figure 9.27. As can be seen, usage of the disruptive C-XTD crossover operator results in many small scattered land-use clusters. The solution clearly has a low compactness, compared to a solution retrieved by PLS-MOLA as was shown previously in Figure 9.18. The PLS-MOLA solution also shows more similarities with the current situation, whereas the global NSGA-II-MOLA algorithm only shows a few. In case it is desired to examine local optima that are close to the current situation, PLS-MOLA would therefore be recommended over NSGA-II-MOLA as well.

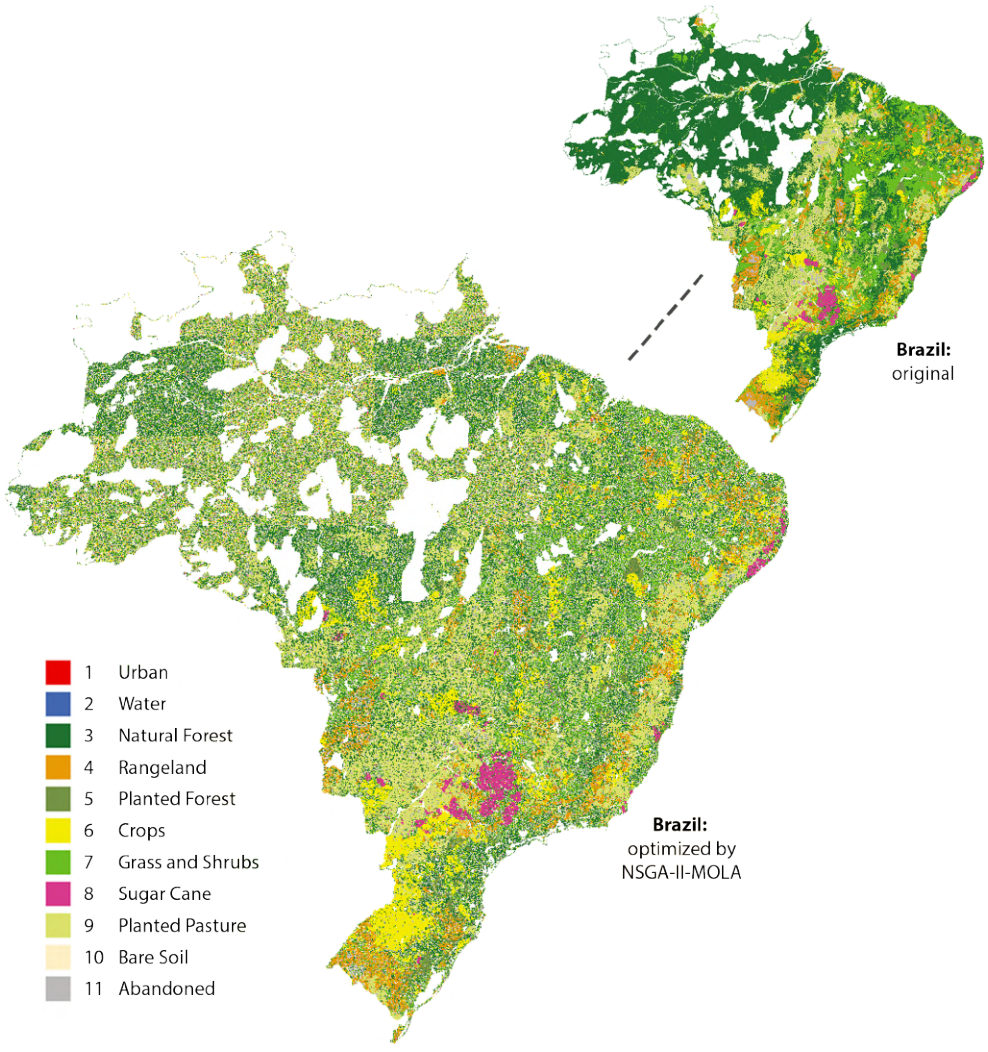


FIGURE 9.27: Brazil with NSGA-II-MOLA

The next problem case on which NSGA-II-MOLA was ran, is Centre West. The obtained hypervolumes are shown in Table H.13 and statistics regarding the final population are shown in Table H.14. The Centre West case is smaller than the Brazil case as it contains 77.9% less dynamic cells. In Figure 9.28, the

generations are visualised in the same manner as before in Figure 9.26. As can be seen, the convergence of the NSGA-II-MOLA populations over time looks similar to when run on the larger Brazil case. However, the different populations now form more solid fronts, clearly improving over previous generations. The populations of Figure 9.26 were more scattered and merged into each other. The initial population also shows similarities; the randomly generated half again shows yield values comparable to the mutated half, with clearly lower compactness values. However, in contrast to the large Brazil case, follow-up generations were able to not only improve upon the potential yield values, but also on the compactness. As so, the capability of optimizing the compactness seems to increase with smaller problem case sizes. Possibly, this is due to the local operators (mutations) now being able to improve more upon the compactness than the disruptive C-XTD operator can take down (as the problem size is now smaller and the number of local search operations per generation remains the same). As so, it might be interesting to increase the number of local search operations per generation (mutation size) when optimizing larger problem cases.

As the Centre West case was also not eligible for IPLS-MOLA, the NSGA-II algorithm will again be compared to PLS-MOLA. The archives of PLS-MOLA over time are visualized in Figure 9.28 as well. Similar to the Brazil problem case, the final PLS-MOLA archive dominates the NSGA-II-MOLA population. The average potential yield value of the final NSGA-II-MOLA population is 1.6173×10^8 with an average compactness of 0.9964×10^5. The average compactness value of the final PLS-MOLA archive was 1.2520×10^5, with an average potential yield value of 1.8303×10^8. Again, the global NSGA-II-MOLA algorithm is not able to find a global optimum better than the (local) optimum found by PLS-MOLA. Possibly, the crossover operator are still not effective enough given the current problem size. As the increase in objective values decreases over the NSGA-II-MOLA generations, it also does not look like the solutions will improve upon the PLS-MOLA solutions in the near future. The PLS-MOLA algorithm is therefore considered to outperform NSGA-II-MOLA in terms of solution quality on the Centre West case as well.

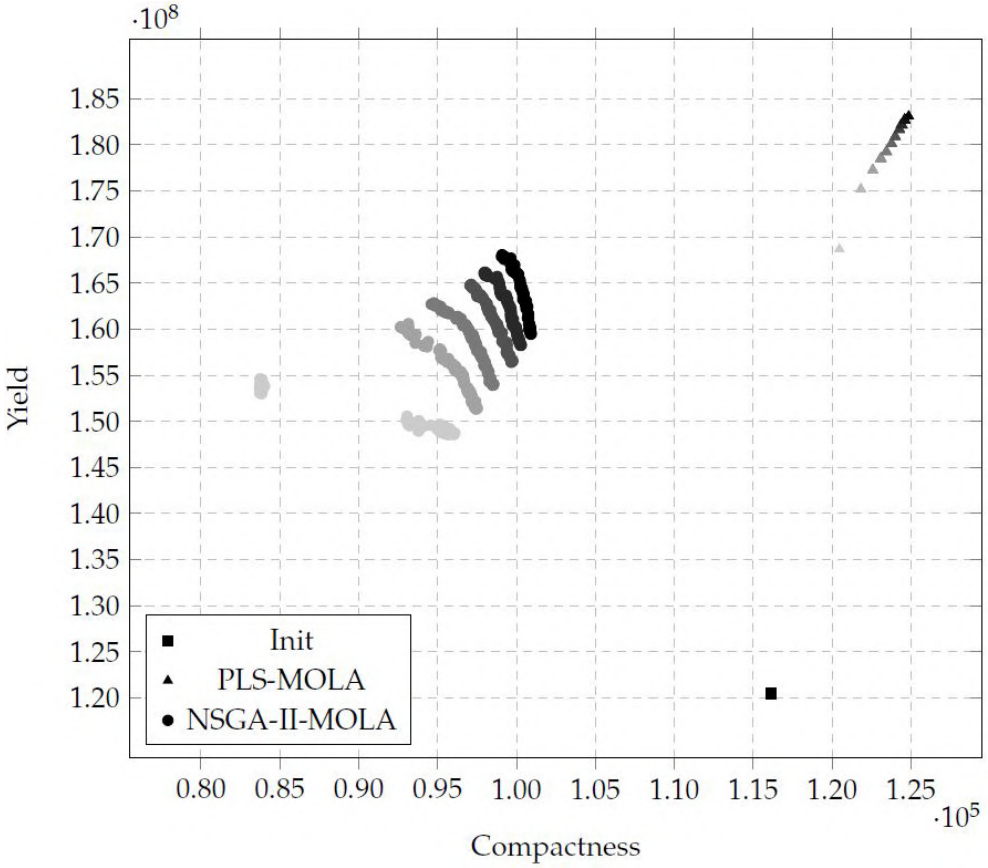


FIGURE 9.28: Convergence of PLS-MOLA and NSGA-II-MOLA for Centre West

One of the solutions of the final population of NSGA-II-MOLA for Macroregoin 4 is visualised in Figure 9.29. Similar to the Brazil case, the solution contains a lot of small clusters, scattered all over the area. Again, this is expected to be caused by the disruptive C-XTD operator. The solutions provided by PLS-MOLA, of which one was shown in Figure 9.20, show larger and more compact clusters. Again, these solutions also show more similarities with the current situation, as is expected with a local search algorithm.

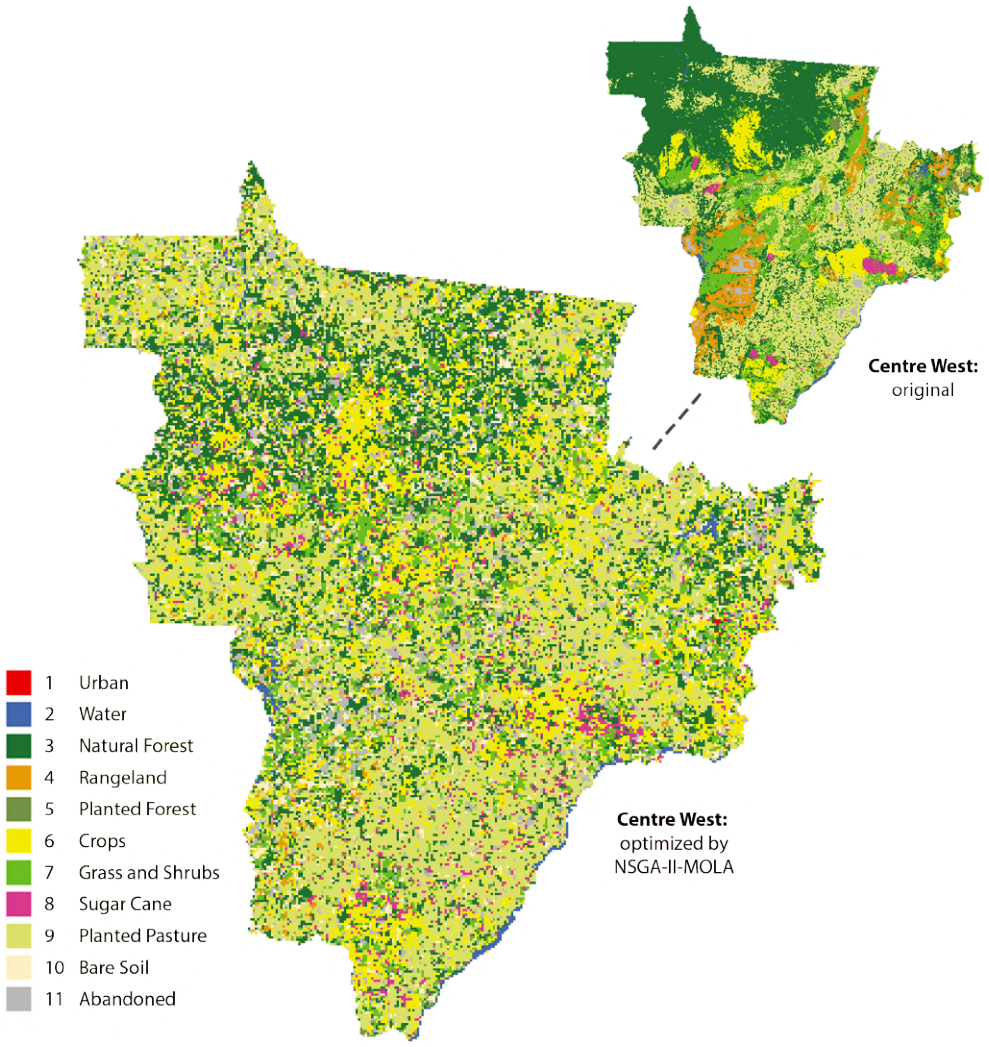


FIGURE 9.29: Centre West with NSGA-II-MOLA

Finally, the NSGA-II-MOLA algorithm was applied to the Sul Goiano case, which is only 1.9% the size of the Brazil case. The obtained hypervolumes are shown in Table H.15 and statistics regarding the final population are shown in Table H.16. The convergence of the populations over time is shown in Figure 9.30. Again, it shows a convergence similar to the one of Brazil and Centre West. Now, let us first compare the performance of the NSGA-II-MOLA algorithm to the local PLS-MOLA algorithm. The PLS-MOLA algorithm finds a local optimum with an average compactness of 7.800×10^3 and an average potential yield value of 1.447×10^7 (Table H.6). For the first time, the population of NSGA-II-MOLA is therefore not dominated by the local PLS-MOLA algorithm. However, note that the NSGA-II-MOLA hereby ran for a much longer time. This eventually resulted in a final population with an average potential yield value of 1.4832×10^8 and an average compactness of 7.055×10^3. Therefore, the global NSGA-II-MOLA is able to improve on the potential yield values of the final PLS-MOLA archive. The global search therefore seems to have found an optimum with a more promising potential yield than the local optimum found by PLS-MOLA. The solutions of PLS-MOLA still have higher compactness values, more likely caused by the application of more local search operations (such as LS-KRBM).

However, the local PLS-MOLA algorithm can be transformed into an IPLS-MOLA algorithm, making it a global search algorithm. As discussed previously in Chapter 9.4, a perturbation size of 25 resulted in the highest hypervolume for the Sul Goiano case (Table H.7). As so, the IPLS-MOLA was run on the Sul Goiano case with a perturbation size of 25, of which the results can be found in Table H.10. The archives over time are shown in Figure 9.30. As can be seen, the IPLS-MOLA algorithm was successfully able to perturb solutions from the local optimum of PLS-MOLA, and find new optima with especially higher potential yield values. The final archive had an average compactness of 7.738×10^3 and potential yield value of 1.569×10^7. All solutions of the final archive hereby dominate the solutions found by NSGA-II-MOLA. Again, it is difficult to estimate whether a longer running time would result in NSGA-II-MOLA outperforming IPLS-MOLA. If we look at all of the three cases previously discussed, the NSGA-II-MOLA solutions come closer to the archive solutions (in terms of objective values) as the problem size decreases. When further decreasing the problem size, there might be a turning point where NSGA-II-MOLA performs equally as good or even better than IPLS-MOLA. This would be an interesting future research direction, and will be discussed further in Chapter 11. However, considering cases the size of Sul Goiano or larger, the NSGA-II-MOLA is currently not able to compete with IPLS-MOLA.

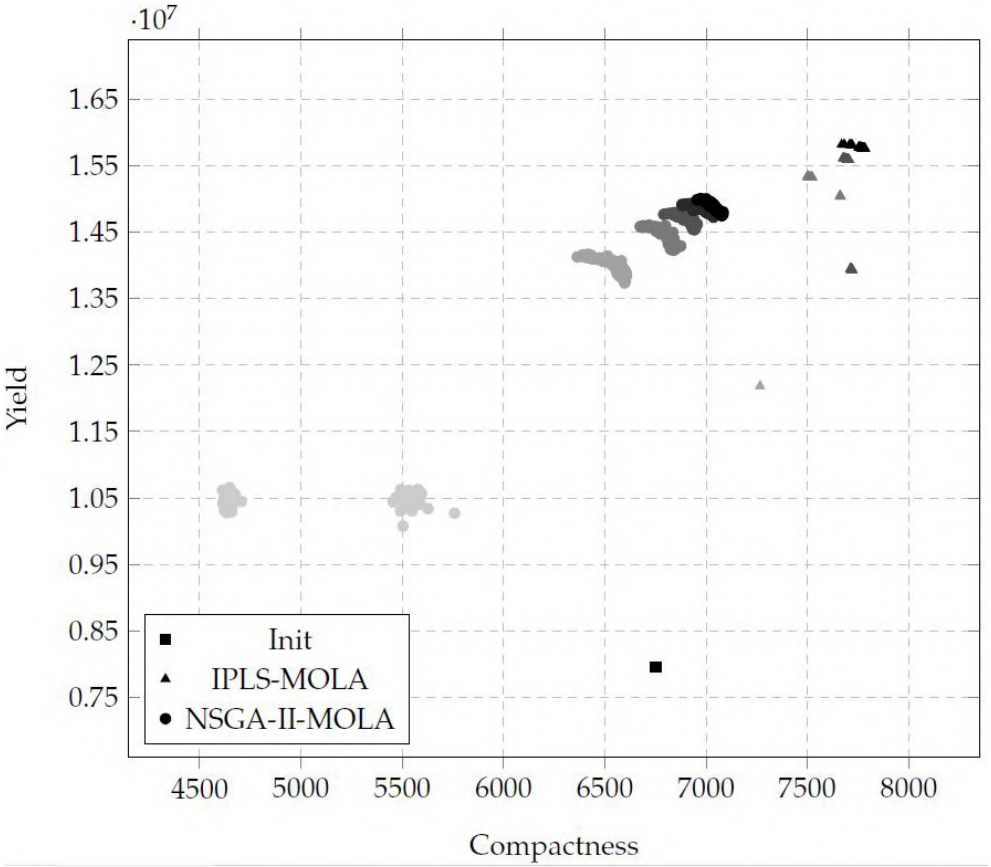


FIGURE 9.30: Convergence of IPLS-MOLA and NSGA-II-MOLA for Sul Goiano

In a final attempt to improve the performance of the NSGA-II-MOLA algorithm, a few more experiments were conducted in which the initial map of

Sul Goiano was included in the initial population. In previous runs, the initial population only contained mutated versions of the initial solution. However, as the algorithm seems to have difficulties effectively optimizing the compactness, addition of the relatively compact initial map to the initial population might aid in the optimization process. Therefore, NSGA-II-MOLA was run 5 more times on the Sul Goiano with the same parameter settings as before, but with the (repaired) Sul Goaino map added directly to the population. The resulting hypervolumes and final population statistics can be found in Table H.17 and H.18. A comparison between the hypervolumes is shown in Figure 9.31. As can be seen, the algorithm now starts off with a higher hypervolume than before. However, as more populations pass, the difference in hypervolume for NSGA-II-MOLA with and without the initial map in the population decreases. One might expect the hypervolume of the initial population with the initial map to be higher; however, repairing the initial solution (with the current settings) still decreases the compactness of the initial solution to around 6200. For more details on the convergence, view Figure H.1 in Appendix H. Eventually, the final average hypervolume obtained when running NSGA-II-MOLA with the initial population is 1.06454×10^{11}, which is slightly larger than the average hypervolume of 1.06433×10^{11} obtained when ran without it. A two-tailed independent sample t-test on the hypervolumes results in a p-value of 0.920744. Also, comparing the compactness values only returns a p-value of 0.940949, with 0.87641 for the potential yield values. As all are above 0.05, we can state that running NSGA-II-MOLA with and without the initial map in the initial population does not significantly influence the solution quality. It is expected that the same holds for the larger Brazil and Mesoregion 4 case. As so, this experiment will not be executed with the other case studies as well.

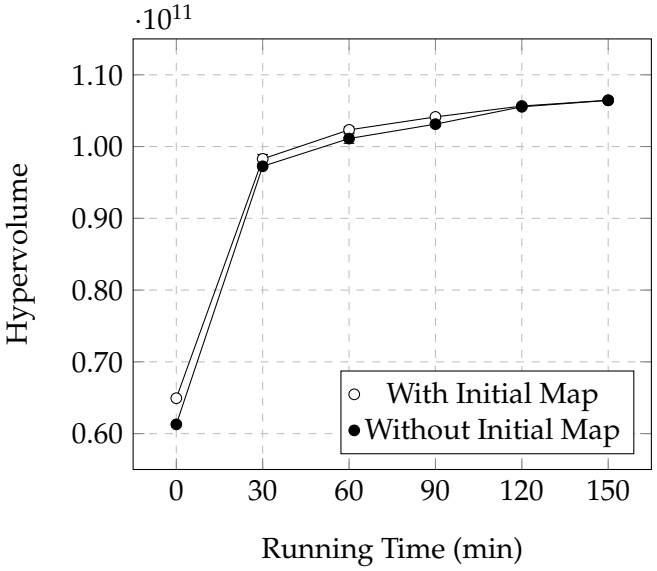


FIGURE 9.31: Hypervolume versus Running Time for NSGA-II-MOLA with and without the Initial Map of Sol Goiano in the Initial Population

Finally, one of the solutions of the final population of NSGA-II-MOLA is shown in Figure 9.32. Compared to solutions of NSGA-II-MOLA for larger cases, the solution is relatively compact, with a few scattered clustered. Possibly, this size allowed the local search operators (mutations) to cope better

with disruptive operations. As expected, a PLS-MOLA solution is still the most compact (Figure 9.22), followed up by an IPLS-MOLA solution (Figure 9.25).

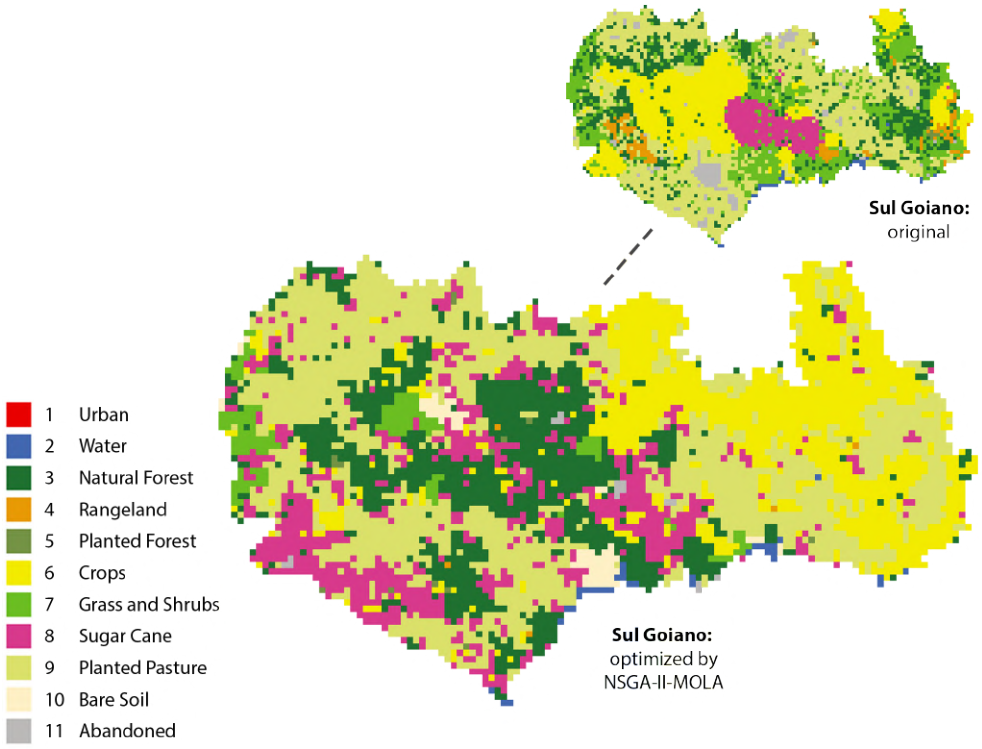


FIGURE 9.32: Sul Goiano with NSGA-II-MOLA

Chapter 10

Conclusions

Multi-objective land-use allocation (MOLA) is a spatial optimization problem in which different land-use types need to be allocated to a set of land units, subjecting to multiple objectives and constraints. MOLA can be found in many research domains and fields of applications, such as urban planning and sustainable societal growth and development. MOLA is a combinatorial optimization problem that can have a significantly large solution space. It is considered to be an NP-hard problem. Over the previous decades, multiple techniques have been applied to solve MOLA problems. Earlier attempts tried to rephrase the MOLA problem into a single-objective optimization problem, whereas later more popular techniques apply multi-objective search heuristics to approximate the complete Pareto front in one execution. The currently most applied technique, is the multi-objective genetic algorithm NSGA-II.

However, the NSGA-II algorithm has multiple drawbacks. Designing efficient crossover operators for MOLA can be difficult, and the disruptive operators currently being used can require many repair operations to maintain valid offsprings. Also, resulting solutions can deviate significantly from the current land-use situation, which is not always desired. Searching in the neighborhood of the current situation can be enforced with an extra objective, but doing so decreases the optimization efficiency of the other objectives (as we will discuss later). As so, an alternative multi-objective algorithm was proposed for the optimization of MOLA problems; Pareto Local Search (PLS). In PLS, an archive of solutions is constantly improved, by exploring the solutions' neighborhoods. With effective operators, this could lead to a more controlled search with less constraint violations as in NSGA-II. Whereas PLS can be used for finding local optima near the current situation, Iterated PLS (IPLS) can be used for a global search in the whole search space. In IPLS, solutions are perturbed to escape local optima and find land-use allocations strongly deviating from the status quo as well.

Throughout this research, a modular PLS and IPLS algorithm have been designed for MOLA, referred to as PLS-MOLA and IPLS-MOLA. The algorithms are designed to specifically fit the MOLA problem, exploiting the efficiency of incremental constraint checking and objective value calculation. Besides, a new data structure has been designed for storing MOLA solutions. The structure stores adjustments to the initial map only, preventing the need of duplicating complete solutions to reduce the memory requirements. Furthermore, three problem dependent local search operators (LS-KRCM, LS-KRBM and LS-KRPM) and two repair operators (LR-KRCM and LR-KRBM) were set up. The operators extend operators of previous research by exploiting domain specific knowledge and are compatible with the proposed data structure. Finally, a modified version of the currently often used NSGA-II

algorithm, referred to as NSGA-II-MOLA, was set up. This modular version also supports the proposed data structure and is able to implement the search and repair operators discussed above.

In order to examine the PLS-MOLA, IPLS-MOLA and NSGA-II-MOLA algorithm, a realistic and topical MOLA problem case has been set up. The problem case was based upon the research of Hilst, Verstegen, Woltjer, et al., 2018, which examined direct and indirect land-use changes resulting from the increasing food, feed and biofuel production in Brazil. Understanding land-use change dynamics resulting from this production, could give insight in the negative effects and sustainability issues that might occur. By converting the case to an optimizable MOLA problem, we could discover how these negative effects can be avoided or minimized in the future. The Brazil problem case contains 278472 dynamic cells, 12 land-use types, one allocation ranges constraint, and three objectives; compactness maximization, potential yield maximization and carbon stock maximization. Eventually, one smaller subcase called Centre West of 63827 dynamic cells and one much smaller subcase called Sul Goiano of 5229 cells was set up. With the last one still being significantly larger than most MOLA cases optimized in previous research, the problem cases can be considered as relatively large MOLA problems.

RQ1 The first research question comprised how the PLS algorithm could most efficiently be applied to MOLA. In order to answer this question, six subquestions were formulated that examine the efficiency of different components, operators criteria and data structures regarding the PLS algorithm. The efficiency is hereby measured in the hypervolume obtained after a certain running time. In case the objectives are being maximized, a higher hypervolume indicates a higher quality Pareto front. In order to answer these questions, several experiments have been conducted. All experiments examining the efficiency of a single parameter or operator were obtained by running PLS-MOLA on Brazil for 5 minutes while measuring the hypervolume. Finally, the complexity of some components was examined by measuring the running time to complete 1000 iterations. The results of the six subquestions together give an indication of how PLS can be set up for MOLA most efficiently.

RQ1.1 The first subquestion concerned the design of a more efficient data structure for MOLA than naively storing the land-use type of each cell in the map. Therefore, the so-called 'initial' and 'derived' solutions were designed; the initial solution stores the land-use type of all cells (naively), whereas the derived solution only stores the adjustments made to an initial solution that it refers to. As some cells in the search space never change, their cell type is now never stored twice, thus decreasing the redundancy. The number of updated cells is always below or equal to the total number of cells on the map, and so this structure will always require less or equal memory space per solution than when stored naively. The PLS algorithm can now store the initial map as an initial solution, and all other solutions generated throughout the optimization process as derived solutions based upon this initial solution. No experiments have been conducted to examine the exact memory space being used throughout the optimization process.

RQ1.2 The second subquestion questioned the influence of the allocation

range size on the running time of PLS-MOLA. The allocation range size is the distance between the lower and upper bound of the number of cells to which a type should be allocated. In order to examine this influence, PLS-MOLA was run on Brazil given different rang sizes for the same number of iterations. The results show that a smaller range size results in a longer running time to complete the same number of iterations. A larger range results in less constraint violations (and so repair operations), which decreases the running time necessary for an iteration. However, this also decreases the accuracy with which a user is able to determine to how many cells each land-use type is allocated. Eventually, it is up to the user to decide what is a suitable trade-off between accuracy and running time for the given situation.

RQ1.3 The third subquestion questioned how to set up an efficient selection strategy. The selection strategy is responsible for what solution is chosen next from the archive, of which the neighborhood will be explored. Therefore, two selection operators where compared; S-R, which randomly selects a solution from the archive, and S-CD, which selects the solution with the highest crowding distance. A higher crowding distance indicates a less extensively explored neighborhood, and is therefore preferred. However, calculating the crowding distances also asks for more operations to be executed. Eventually, there was no significant difference in the hypervolumes (efficiency) resulting from a PLS-MOLA optimization on Brazil using either S-R, S-CD or both.

RQ1.4 The fourth subquestion questioned how to set up an efficient exploration strategy. The exploration strategy is responsible for the exploration of the neighborhood of a solution. First of all, three local search operators have been designed and compared; LS-KRCM, which alters a random cell, LS-KRBM, which alters a random boundary cell, and LS-KRPM, which alters a random patch of 7 cells. Using only LS-KRBM or a combination of all three operators resulted in the highest hypervolumes, with no significant difference. However, as the combination of all three operators is expected to deliver more diverse solutions, one might prefer this set over solely LS-KRBM. Next, the Exploration KT value was examined, which states the number of types competing in the tournament that decides what type the selected cell or patch should become. The one resulting in the highest objective value of one (random) objective is applied. Eventually, the Exploration KT parameter did not significantly influence the resulting hypervolumes. Finally, the Exploration N parameter was examined, which states the number of solutions explored per neighborhood exploration. The results show an exponential reduction in hypervolume as the Exploration N decreases. However, as we are only examining the first 5 minutes of running time, improved solutions are still found easily and so it is indeed efficient to continue to the next iteration fast. However, with longer running times, a different result is expected as low Exploration N values might not be sufficient for the algorithm to find improvements in time. More research using longer running times would therefore be advised to expose this behavior as well. Concluding, low Exploration N values are efficient in the first phase of the optimization algorithm, but more research is required to examine its late time efficiency as well.

RQ1.5 The fifth subquestion questioned how to set up an efficient reparation strategy. The reparation strategy is responsible for repairing solutions that violate the problem constraints. First of all, two repair operators were designed

and compared; LR-KRCM, which repairs a solution by altering cells, and LR-KRBM, which repairs a solution by altering boundary cells. No significant difference was detected in the hypervolumes obtained from running PLS-MOLA on Brazil when using either LR-KRCM, LR-KRBM or both. Again, as the combination of both operators is expected to deliver more diverse solutions, one might prefer using both instead of one. Next, the influence of the Reparation KC parameter was examined. The Reparation KC parameter states the number of (boundary) cells competing in the tournament of one repair operation, in which the cell resulting in the highest value of one (random) objective wins. The results show that the function describing the hypervolume of the repaired initial solution versus the Reparation KC value has a logarithmic form, in which the increase in hypervolume is smaller with higher Reparation KC values. However, the reparation time of the initial solution also rises linearly with the Reparation KC value. As so, a trade-off will have to be made by the user between reparation quality and reparation time. Thereafter, the Reparation BT parameter was examined. If set to true, the search operators always choose the best type (considering one random objective) for a cell or patch (in case multiple types can be applied). Else, this type will be chosen at random. In case the Reparation BT parameter was set to true, the hypervolume resulting from running PLS-MOLA on Brazil was significantly higher than when set to false. This shows that a competition between types has a positive effect on the eventual quality, although more operations are required. Finally, the Allow Repair parameter was examined, which states whether PLS-MOLA should immediately discard invalid solutions that are found (and keep on searching for new ones until a valid one is found) or allow the reparation of invalid solutions. The experiments show that allowing reparations, meaning Allow Repair is set to true, results in significantly higher hypervolumes.

RQ1.6 The sixth and final subquestion questioned what acceptance criteria result in the most efficient optimization process. First of all, the NDS criterion was examined, which states that a new solution may only be added to the archive if it dominates the current solution being explored. Results show, that when this criterion is used, the hypervolume is significantly higher. However, it also shows the number of solutions in the archive is significantly less. This can especially bring difficulties at later stages of the optimization process, when it becomes more difficult to find improvements. As so, the choice of whether the NDS criterion should be used is a trade-off between a faster increasing hypervolume and a larger and more diverse archive (increasing the chance of finding improvements in later stages). The second and final criterion being examined, is the maximum archive size. This criterion states the maximum number of solutions the archive may contain, and makes the archive drop the solution(s) with the lowest crowding distance in case it is violated. No significant differences were found in the hypervolumes obtained when using different maximum archive sizes.

RQ2 The second research question comprises the relation between the number of objectives and the efficiency of PLS for MOLA. As discussed before, three objectives were set up for the Brazil case. For examining RQ1, the problem was optimized with only compactness and potential yield maximization as the objectives. For this research question, the effects of including carbon stock maximization among these objectives was examined as well. First of

all, the average number of solutions in the archive was significantly larger. After 1000 iterations, the PLS-MOLA archive contained (on average) 15.0 solutions when maximizing compactness and potential yield and 1135.4 solutions when carbon stock maximization was added as well. This significant increase is caused due to the fact that a solution is less often dominated with three (real valued) objectives. In later stages the archive can expand even faster, possibly resulting in high memory demands. A maximum archive size could be used to prevent this from happening. Next, the hypervolumes of only compactness and potential yield were compared (neglecting the carbon stock values). When optimizing only these two objective, the hypervolume obtained from running PLS-MOLA on Brazil was significantly larger than with the addition of a third objective. This was expected, as on average more focus is paid to each objective individually by the local search operators. As so, the user should always ask whether the addition of an extra objective is worth this decrease in optimization quality. When optimizing more than three objectives, this decrease is expected to be even higher. As will be discussed in Chapter 11, it could therefore be interesting to look at alternative strategies to include more objectives without increasing the total number of objectives being optimized simultaneously. Finally, an interesting observation showed that the running time to complete 1000 iterations is significantly shorter when carbon stock maximization is added to the objectives. This is caused by the fact, that when optimizing the potential yield, more repair operations are required (on average). As local search operators optimize the potential yield more often when optimizing only two objectives, the eventual number of reparations required is higher and so is the running time. As so, when estimating the influence of an objective on the running time, one should not only look at the costs of calculating and updating the objective itself.

RQ3 The third research question asks how the local PLS algorithm can efficiently be processed into a global IPLS algorithm for MOLA. As the IPLS algorithm iteratively calls the PLS algorithm, the answer to RQ 1 is essential for an efficient IPLS implementation as well. In order to further optimize the global IPLS algorithm, an effective perturbation size needs to be found. If the perturbation size is too small, the IPLS algorithm will not be able to escape a local optimum. If the perturbation size is too large, the perturbed solution loses a lot up to all of its valuable information. An efficient perturbation size will differ per problem case, and therefore this will be examined for each case separately. However, for the Brazil and Centre West case, the PLS-MOLA algorithm did not find (approximate) a local optimum in 2.5 hours. As the search space is extremely large, longer running times or better hardware would be required, which was out of the scope of this research. As so, the IPLS-MOLA algorithm has only been applied to and optimized for the smaller Sul Goiano case. A perturbation size of 25%, 50% and 75% were examined, which state the percentage of dynamic cells that are given a random land-use type. The results show, that a perturbation size of 25% resulted in a significantly higher final hypervolumes (than both 50% and 75%) for the Sul Goiano case. As the final hypervolume seems to increase with smaller perturbation sizes, it might be interesting to examine perturbation sizes of below 25% for the Sul Goiano case in the future as well.

RQ4 The final research question comprises how the performance of the (I)PLS

algorithm compares to the performance of NSGA-II algorithm when applied to MOLA. In order to do so, a modular NSGA-II algorithm was designed, called NSGA-II-MOLA. This modality allows the algorithm to combine components and operators found most effective in either this or previous research. As so, the crossover operators and initialization procedure are based upon previous research, whereas the repair and local search operators are based upon this research. This includes the possibility to reuse the parameter settings found most effective for RQ 1. Finally, the NSGA-II-MOLA algorithm also exploits the same data structure for MOLA as used in (I)PLS-MOLA. In this manner, the performance of the NSGA-II algorithm for MOLA was optimized for as far as possible within the scope of this research.

At first, the performance of NSGA-II-MOLA was compared to the performance of PLS-MOLA for the Brazil case (278472 dynamic cells) and Centre West case (63827 dynamic cells) case. Each algorithm was run for 2.5 hours on each case for five times, of which the interim and final populations/archives were stored. The solutions resulting from NSGA-II-MOLA improved the potential yield value of Brazil on average with 27.7%, while decreasing the compactness with 15.6%. Solutions of PLS-MOLA improved the potential yield value with 39.9% and the compactness on average with 3.2%. The second case showed similar results, with NSGA-II-MOLA improving the potential yield with 34% while decreasing the compactness with 14.2% and PLS-MOLA improving the potential yield with 51.9% and compactness with 7.8% (all on average). In both cases, all solutions in the final archive of PLS-MOLA dominate the final population of NSGA-II-MOLA. The results imply, that the crossover operators of NSGA-II-MOLA do not allow for an efficient exploration of the relatively large search space. The disruptive C-XTD operator can cause large changes, which could again require a large number of repair operations to fix the resulting offsprings. Especially the compactness of the offsprings does not seem to improve effectively. The current amount of local search is also not able to compensate for this loss. Designing crossover operator that better preserve the fit genes without causing many violations, might help NSGA-II-MOLA to overcome these difficulties. Also, the amount of local search could be increased, although this should not be essential for realizing an effective optimization process. The local PLS-MOLA algorithm on the other hand, continuously seems to succeed in improving both the potential yield and compactness of the solutions in the archive. The local search operators effectively find new higher quality neighboring solutions, causing a steady process that incrementally improves the average solution quality. No local optima were found (approximated) in the 2.5 hours running time, due to the large search spaces. Overall, the PLS-MOLA algorithm clearly outperforms the NSGA-II-MOLA algorithm in terms of solution quality, proving to be the better option for MOLA problems the size of Centre West or Brazil.

Finally, the performance of NSGA-II-MOLA was compared to the performance of IPLS-MOLA for the Sul Goiano case. This case only contained 5229 dynamic cells, which was small enough for PLS-MOLA to reach (approximate) a local optima within 2.5 hours. Again, five runs of 2.5 hour with IPLS-MOLA (with the perturbation size set to 25%) and NSGA-II-MOLA were performed. The resulting population of the NSGA-II-MOLA showed

an average increase in the potential yield of Sul Goiano of 86.4% and an increase in compactness of 4.5%. NSGA-II-MOLA is now finally able to improve upon the compactness of the initial solution. Also, the global NSGA-II-MOLA algorithm improves upon the potential yield of local optimum found by PLS-MOLA. However, when the global IPLS-MOLA is run for the same amount of time, the NSGA-II-MOLA algorithm is still outperformed. The resulting archive of IPLS-MOLA showed an average increase in potential yield of 97.2% and an increase in compactness of 14.6%. Again, all solutions of NSGA-II-MOLA were being dominated by the final archive. In an attempt to improve upon the optimization efficiency of NSGA-II-MOLA, five more runs were performed in which the (repaired) initial map was added directly to the initial population. However, this resulted in no significant improvements in the final hypervolume obtained after 2.5 hours. Therefore, even for smaller cases as Sul Goaino, PLS-MOLA is proven to be the best performing algorithm in terms of solution quality. Finally, the user might desire to only search for solutions that are similar to the current situation, to for instance examine how the current situation can be expended best. Optionally, this can be enforced with an extra objective and constraint, but as shown for RQ 3 adding more objectives is generally undesirable. However, when we look at the solutions resulting from PLS-MOLA, IPLS-MOLA and NSGA-II-MOLA, the solutions of PLS-MOLA algorithm already automatically show a lot of resemblance with the current situation (as is expected with a local search algorithm). Therefore, PLS-MOLA can be an efficient option (obviating the need of more objectives and constraints) when optimizing solutions similar to the current situation only is desired.

Research Goal The overall goal of this exploratory research project was to examine the potential of PLS in the MOLA domain. One of the biggest challenges that current MOLA techniques are facing, is to obtain a high scalability. This also applies to the currently most used MOLA technique, namely NSGA-II, of which the current crossover operators have difficulties in exchanging genes efficiently without causing many constraint violations. The PLS algorithm was presumed not to have this issue, as taking small local steps in the search space can more easily be done without any constraint violations, decreasing the need of repair operations and so increasing the efficiency and scalability. Therefore, the overarching research question "Is PLS an efficient algorithm for MOLA?" was set up. In order to answer this research question, PLS needed to be adapted to and optimized for MOLA and the efficiency needed to be compared to current state-of-the-art algorithm(s). In case the efficiency is equal to or higher than the efficiency of those algorithms, the PLS algorithm will be considered as 'efficient' for MOLA. Therefore, RQ 1 to 4 were set up and PLS-MOLA and IPLS-MOLA were implemented, optimized and compared to NSGA-II. The algorithms were applied to three topical MOLA problems of a relatively large size and run for 2.5 hours to optimize two different objectives. The results show that the solutions of (I)PLS-MOLA dominate the solutions of NSGA-II in all cases, meaning that the obtained hypervolumes are larger at all time. Given the current running times, (I)PLS therefore proves to be more efficient than NSGA-II for all three MOLA cases. The suspicion that PLS would be able to optimize MOLA problems more efficiently than NSGA-II, therefore seems to be confirmed. Taking smaller and more controllable steps in the enormous search space of these MOLA

problems can indeed lead to a more efficient approach than applying disruptive crossover operators that more easily violate the problem constraints. Especially for larger cases, (I)PLS brings significant improvements in solution quality, and so proves to be an interesting option for future MOLA optimization systems. To come back to the overarching research question; PLS shows to be an efficient algorithm for MOLA compared to current state-of-the-art MOLA techniques. Although its behavior has not yet been examined in smaller cases, the results in cases the size of Sul Goiano and larger are promising. However, this was only an exploratory research project and more research will be required to further examine and exploit the possibilities of this technique. This will be discussed in more detail in Chapter 11.

Limitations Although the results indicate that PLS is an efficient optimization technique for MOLA, there are still several limitations and questions that remain unanswered. Part of these form the basis of the recommended research directions that will be proposed in Chapter 11. First of all, regarding the PLS algorithm, the influence of the termination criteria of the neighborhood exploration (Exploration N parameter) on the approximation quality is unknown. A low Exploration N value speeds up the convergence but could result in a lower final approximation quality due to the PLS algorithm terminating more easily. One way to overcome this issue, is by introducing a different criterion as will be suggested in Chapter 11. Secondly, a new data structure was designed for the PLS algorithm to reduce the amount of memory space required to store a MOLA solution. This decrease in required memory space was substantiated with reasoning, but the actual reduction in memory space (over time) is yet unknown. Next, when looking at the NSGA-II algorithm that has been applied to MOLA, not all operators and parameters have been optimized. The algorithm implements multiple components and operators that have proven to be efficient in previous and current research, but still there were several parameters (for instance the population size and composition of the initial population) that were out of the scope of this research to optimize. Currently, it is unknown to what extent an optimization of NSGA-II would improve upon its performance. Finally, the experiments were applied to one MOLA case (and two subcases) only. As we would like to gain insight in how the efficiency of PLS relates to NSGA-II and other techniques in general, it will be necessary to examine the algorithms given other cases and compared to other algorithms as well. This includes case studies of a smaller size than the Sul Goiano case. Also, the running time and hardware used, could be increased to gain insight in later optimization stages as well. In Chapter 11, several of these limitations will be translated into potential future research directions.

Chapter 11

Recommendations

In this research, the first steps have been made in the exploration of the possibilities of PLS and IPLS for MOLA. Previously to this research, the PLS algorithm had not yet been applied to this optimization problem. As so, the first version of a PLS algorithm for MOLA has been designed, optimized and compared to a current state-of-the-art MOLA technique. The results show that PLS is able to bring multiple advantages and proves to be an interesting future research direction. Especially for large sized MOLA problems and optimizations in which it is desired that the optimized solution does not deviate too much from the current situation, PLS can bring significant value. In order to fully exploit its potential, more future research will be required. For example, to get more insight in the behavior of PLS when applied to smaller study cases or executed for longer running times. In this chapter, seven future research directions will be outlined that are considered to be interesting based on the results of this research. A direction is considered as interesting, in case it is expected to give new valuable insights or improvements for either PLS, IPLS or NSGA-II for MOLA.

Exploration Criteria. The first research direction concerns a possible enhancement of PLS and IPLS. In Chapter 9.2.4, the exploration strategy of PLS was examined for the answering of RQ 1C. One aspect of the exploration strategy is the termination criterion used for the neighborhood exploration. Currently, this termination criterion was provided with the Exploration N parameter, which comprised the number of neighborhood solutions to be explored after which the neighborhood exploration ends. The results (Table G.7 and Figure 9.5) of Run 35 to 41, showed an increase in hypervolume as the Exploration N decreases. For the first 5 minutes, this is indeed the case, as 'improved' solutions are still easy to find in a neighborhood. A low Exploration N causes the algorithm to quickly move on to the next iteration, eventually resulting in a higher hypervolume. However, later on in the optimization process, improvements are harder to find. As so, a low Exploration N could result in an earlier termination of the algorithm when no better solutions are found in time. As so, the Exploration N is a trade-off between optimization speed at the start of the process and the ability to find improvements in later stages. This is undesirable and can be avoided, by simply using the number of solutions added to the archive as the termination criterion. For instance, when for set to 10, the neighborhood exploration of a solution will terminate when 10 new neighbor solutions have been added to the archive. This will result in both a fast convergence at the start, and the ability to continue on improving in later stages. Therefore, implementing this termination criterion and examining its influence on the convergence speed and quality could be an interesting future research direction. Note, that the Exploration N criterion is still necessary in order to terminate the algorithm when a local optimum is found or closely approximated.

Adaptive Procedures. The current PLS algorithm implements a selection procedure, search procedure and repair procedure (Chapter 7). These procedures are responsible for selecting what corresponding operators will be chosen and applied next. For example, when the search procedure is called, it selects and applies one operator from the set of search operators that was provided to the algorithm. The selection and repair procedure do the same for selection and reparation operators. Currently, the procedures chose the next operator to be used at random. As so, each operator has an even chance of being chosen. However, in some cases it might be more efficient to select one operator more often than another operator. First of all, this can vary throughout the optimization process; one operator can be efficient in the starting phase, whereas another operator is more efficient in later stages. Secondly, this can vary per problem; some operators might be effective on the map of Brazil, whereas others are more efficient when used on Sul Goiano. In order to overcome this 'problem', adaptive procedures could be implemented. The procedures could take note of how efficient each operator (during each part of the process) is, and continuously adapt the chance of each operator to be selected based on this efficiency. Doing so, could allow the user to pass on a diverse set of operators to the algorithm, which decides by itself which of these will be used most. As so, examining the efficiency of operators manually, as was done in Chapter 9.2.4, would become redundant. Implementing these adaptive procedures and examining their effectiveness, could therefore be an interesting future research direction.

Memory Optimization. In order to efficiently store MOLA solutions, a new data structure was designed. The data structure was described in Chapter 7.2.2 and worked as follows; a solution was either an 'initial' solution, containing the land-use type of each cell on the map, or a 'derived' solution, which refers to an initial solution and only stores the adjustments made. As so, it is not necessary to duplicate solutions each time a local search operator is applied. In this manner, the amount of storage necessary to save all solutions in the archive (and the time to create new solutions) is reduced. The current implementation of PLS-MOLA keeps on creating new derived solutions throughout the optimization process, all based on the initial solution of the initial map. However, the total number of adjustments made to the initial solution increases over time. In later stages, derived solutions can therefore still become relatively large. At some point in time, it might therefore be beneficial to store all similarities between the derived solutions in the initial solution. Meaning, that when all derived solutions in the archive have the same land-use type allocation to a certain cell, this type can be stored in the initial solution and removed from all derived solutions. As it is likely that the derived solutions will show some similarities in later stages, this could decrease the redundancy among those solutions and bring down the required memory space. Especially for large problems and PLS-MOLA runs with long running times, this might make a difference. Implementing and researching when and how this storage procedure could best be executed, could therefore be an interesting future research direction.

Problem Scalability. In this research, three case studies for MOLA were set up and examined. The largest case study was Brazil with 278472 dynamic cells. The Centre West case was the second largest, covering 22.9% of Brazil,

and Sul Goiano was the smallest, covering only 1.9% of Brazil. Most case studies used in previous research, including all ten cases discussed in Chapter 3.6.1, are still of a smaller size than Sul Goiano. With the algorithms ran on Brazil and its sub cases, a general impression was given of the scalability of PLS, IPLS and NSGA-II for MOLA. In those relatively large cases, the PLS and IPLS algorithm seem to be the most efficient. This could be expected, as the PLS-MOLA and IPLS-MOLA algorithms have been built with scalability in mind. However, the current research does not provide any insight in the behavior of these algorithms when run on cases smaller than Sul Goiano. It is expected that the IPLS-MOLA algorithm will still outperform NSGA-II, as the advantages of the algorithm compared to NSGA-II remain. However, their relative differences in performance might decrease as, for instance, the decrease in compactness caused by the crossover operators can more easily be covered up with the use of local search. Furthermore, the application of IPLS-MOLA to smaller and more often used problem cases, would allow for a comparison with the results obtained in other researches. As so, examining PLS-MOLA and IPLS-MOLA given smaller cases would be an interesting future research direction to get a better understanding of these algorithms' behavior in smaller search spaces as well.

Increasing Resources. The current research project only had a limited amount of resources available, both in terms of time and hardware. First of all, the experiments were run up to a maximum of 2.5 hours, due to the relatively short time available to execute all experiments. Unfortunately, the PLS algorithm did not reach an optimum within 2.5 hours on the Brazil and Centre West case, making these cases not eligible for IPLS in this research. Increasing the running times up to a time that would allow PLS to reach an optimum at least once (and preferably more often), would make it possible to run IPLS on these cases as well. Although the PLS algorithm did already outperform the NSGA-II algorithm in both cases, meaning IPLS will do so as well, it would still be interesting to gain insight in the convergence of IPLS for these larger cases. Besides increasing the resources in terms of time, it is also possible to improve upon the hardware being used. Currently, the experiments were performed on a Microsoft Surface Pro 7 of which the technical specifications are listed in Chapter 8.4. Improving upon the specifications of the hardware, could allow the algorithms to perform the same tasks in a shorter timespan. Also interesting, is that the algorithms execute the exploration of a neighborhood or creation of offsprings in parallel, as was discussed in Chapter 7.5). In some cases, these tasks can be executed on a graphics card, which can significantly enhance the performance. By increasing the running time and/or improving the hardware, the algorithms will be able to converge beyond the stages reached in this research. As it is likely that more resources will also be available when these techniques are eventually applied to MOLA problems, for example by land-use planners, examining these aspects in future research can provide more insight in what can actually be expected and obtained.

Combining Objectives. The second research question examined the scalability of PLS-MOLA regarding the number of objectives being optimized. The majority of the experiments that were executed throughout this research, involved two objectives; compactness and potential yield maximization. These objectives were regarded as the most practical objectives, and so formed the

basis of the experimental setup. In Chapter 9.3, the effects of adding the carbon stock maximization objective to this set of objectives was examined. First of all, when optimizing three objectives instead of two, the archive size increased significantly. Especially when optimizing large MOLA cases, such as Brazil, this might lead to a large increase in the required amount of memory space. Secondly, the optimization quality of the individual objectives decreased, as the focus of the local search operators had to be divided. It is therefore advised to keep the number of objectives being optimized as low as possible. In case it is still desired to optimize three or more objectives at once, the user could perform several tricks to do so without further increasing the number of objectives being optimized simultaneously. The first possibility is to combine multiple objectives into one. For example, by adding up the objective values in case they fit together. The number of objectives will not be increased, while the algorithm will still take both objectives (although not separately) in account. Another option, is to execute and combine multiple runs. For example, Brazil could first be optimized on compactness and yield, of which a resulting solution is than used as the input of a run that optimizes compactness and carbon. However, it is currently unclear how the optimization quality would be influenced by either one of these tricks. As it is not uncommon that there are more than two objectives to be optimized in real cases, this would make an interesting future research direction. Finally, objectives can also be combined while already being optimized to enhance the overall performance. For example, by giving different priorities (multiplication factors) to the potential yield values of different land-use types, more compactness can be created. The potential yield is often higher in certain areas, and so it will now become more optimal to fill these areas with high priority land-use types only. Especially for algorithms that have difficulties with optimizing compactness, such as NSGA-II, this could be a valuable future research direction.

Enhancing NSGA-II-MOLA. Finally, the NSGA-II-MOLA algorithm was set up, combining different strategies and operators that have proven to be efficient in this and previous research. However, it was out of the scope of this research to optimize NSGA-II-MOLA in the same manner as was done with PLS-MOLA. This includes, examining what parameters would result in the most optimal results (hypervolumes) concerning either one of the cases. As so, the influence of the population size or composition of the initial population on the performance are currently unknown. Furthermore, the current crossover operators are not designed to be applied to large cases such as Brazil. Previous researches, in which these operators were designed, only applied them to cases smaller than Sul Goiano. Possibly, more efficient crossover operators could be created, that are also able to efficiently exchange fit genes when applied to large case studies such as Brazil. Finally, the amount of local search applied during the NSGA-II optimization process can have a huge influence on the convergence of the algorithm. Currently, a small amount of local search was applied to each new offsprings and during the reparation phases. By increasing the amount of local search, the algorithm might be able to perform better in terms of compactness, which was currently one of the main challenges of the algorithm when applied to large cases. Therefore, in the search to the most efficient algorithm for MOLA, a further improved NSGA-II-MOLA algorithm could still be an interesting option.

In summary, seven interesting future research directions have been recommended. The first four recommend directions involving possible improvements for the PLS-MOLA. The fifth and sixth recommend researching the application of the algorithm to other sized cases or with more resources available. The final direction involves several research possibilities for further improving the NSGA-II-MOLA algorithm. The research was performed in cooperation with the Energy and Resources group of Utrecht University, which will continue with this research hereafter.

Bibliography

Abraham, A. and L. Jain (2005). "Evolutionary multiobjective optimization". *Evolutionary Multiobjective Optimization*. Springer, pp. 1–6.

Aerts, Jeroen, Marjan Van Herwijnen, Ron Janssen, and Theodor Stewart (2005). "Evaluating spatial design techniques for solving land-use allocation problems". *Journal of Environmental Planning and Management* 48.1, pp. 121–142.

Aerts, Jeroen CJH, Erwin Eisinger, Gerard BM Heuvelink, and Theodor J Stewart (2003). "Using linear integer programming for multi-site land-use allocation". *Geographical analysis* 35.2, pp. 148–169.

Aerts, Jeroen CJH, Marjan van Herwijnen, and Theodor J Stewart (2003). "Using simulated annealing and spatial goal programming for solving a multi site land use allocation problem". *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, pp. 448–463.

Aerts, Jeroen CJH and Gerard BM Heuvelink (2002). "Using simulated annealing for resource allocation". *International Journal of Geographical Information Science* 16.6, pp. 571–587.

Amine, Khalil (2019). "Multiobjective Simulated Annealing: Principles and Algorithm Variants". *Advances in Operations Research* 2019.

Angel, Eric, Evripidis Bampis, and Laurent Gourvés (2004). "Approximating the Pareto curve with local search for the bicriteria TSP (1, 2) problem". *Theoretical Computer Science* 310.1-3, pp. 135–146.

Cabrera-Guerrero, Guillermo, Andrew J Mason, Andrea Raith, and Matthias Ehrgott (2018). "Pareto local search algorithms for the multi-objective beam angle optimisation problem". *Journal of Heuristics* 24.2, pp. 205–238.

Cao, Kai, Michael Batty, Bo Huang, Yan Liu, Le Yu, and Jiongfeng Chen (2011). "Spatial multi-objective land use optimization: extensions to the non-dominated sorting genetic algorithm-II". *International Journal of Geographical Information Science* 25.12, pp. 1949–1969.

Cao, Kai, Bo Huang, Shaowen Wang, and Hui Lin (2012). "Sustainable land use optimization using Boundary-based Fast Genetic Algorithm". *Computers, Environment and Urban Systems* 36.3, pp. 257–269.

Cao, Kai and Xinyue Ye (2013). "Coarse-grained parallel genetic algorithm applied to a vector based land use allocation optimization problem: the case study of Tongzhou Newtown, Beijing, China". *Stochastic Environmental Research and Risk Assessment* 27.5, pp. 1133–1142.

Czyżżak, Piotr and Adrezej Jaszkiewicz (1998). "Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization". *Journal of Multi-Criteria Decision Analysis* 7.1, pp. 34–47.

Datta, D., K. Deb, C. Fonseca, F. Lobo, P. Condado, and J. Seixas (2007). "Multi-objective evolutionary algorithm for land-use management problem". *International Journal of Computational Intelligence Research* 3.4, pp. 371–384.

Deb, Kalyanmoy, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". *IEEE transactions on evolutionary computation* 6.2, pp. 182–197.

Drugan, Madalina M and Dirk Thierens (2010). "Path-guided mutation for stochastic Pareto local search algorithms". *International Conference on Parallel Problem Solving from Nature*. Springer, pp. 485–495.

Drugan, Mădălina M and Dirk Thierens (2012). "Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies". *Journal of Heuristics* 18.5, pp. 727–766.

Dubois-Lacoste, Jérémie, Manuel López-Ibáñez, and Thomas Stützle (2015). "Anytime pareto local search". *European journal of operational research* 243.2, pp. 369–385.

Duh, Jiunn-Der and Daniel G Brown (2007). "Knowledge-informed Pareto simulated annealing for multi-objective spatial allocation". *Computers, Environment and Urban Systems* 31.3, pp. 253–281.

Flavell, RB (1976). "A new goal programming formulation". *Omega* 4.6, pp. 731–732.

Geiger, Martin Josef (2008). "Foundations of the Pareto iterated local search metaheuristic". *arXiv preprint arXiv:0809.0406*.

Geiger, Martin Josef (2011). "Decision support for multi-objective flow shop scheduling by the Pareto iterated local search methodology". *Computers & industrial engineering* 61.3, pp. 805–812.

Heydari, Majeed and Amir Yousefli (2017). "A new optimization model for market basket analysis with allocation considerations: A genetic algorithm solution approach". *Management & Marketing* 12.1, pp. 1–11.

Hilst, Floor van der, Judith A Verstegen, Geert Woltjer, Edward MW Smeets, and Andre PC Faaij (2018). "Mapping land use changes resulting from biofuel production and the effect of mitigation measures". *GCB Bioenergy* 10.11, pp. 804–824.

Huang, Kangning, Xiaoping Liu, Xia Li, Jiayong Liang, and Shenjing He (2013). "An improved artificial immune system for seeking the Pareto front of land-use allocation problem in large areas". *International Journal of Geographical Information Science* 27.5, pp. 922–946.

Li, X. and L. Parrott (2016). "An improved Genetic Algorithm for spatial optimization of multi-objective and multi-site land use allocation". *Computers, Environment and urban systems* 59, pp. 184–194.

Liu, Xiaoping, Xia Li, Xun Shi, Kangning Huang, and Yilun Liu (2012). "A multi-type ant colony optimization (MACO) method for optimal land use allocation in large areas". *International Journal of Geographical Information Science* 26.7, pp. 1325–1343.

Liu, Yaolin, Jinjin Peng, Limin Jiao, and Yanfang Liu (2016). "PSOLA: A heuristic land-use allocation model using patch-level operations and knowledge-informed rules". *PloS one* 11.6, e0157728.

Martinelli, Luiz A, Rosamond Naylor, Peter M Vitousek, and Paulo Moutinho (2010). "Agriculture in Brazil: impacts, costs, and opportunities for a sustainable future". *Current Opinion in Environmental Sustainability* 2.5-6, pp. 431–438.

Masoomi, Zohreh, Mohammad Sadi Mesgari, and Majid Hamrah (2013). "Allocation of urban land uses by Multi-Objective Particle Swarm Optimization algorithm". *International Journal of Geographical Information Science* 27.3, pp. 542–566.

Masoumi, Zohreh, Jamshid Maleki, Mohammad Sadi Mesgari, and Ali Mansourian (2017). "Using an evolutionary algorithm in multiobjective geographic analysis for land use allocation and decision supporting". *Geographical Analysis* 49.1, pp. 58–83.

Matthews, Keith B, Susan Craw, Stewart Elder, Alan R Sibbald, and Iain MacKenzie (2000). "Applying genetic algorithms to multi-objective land use planning". *Proceedings of the 2nd annual conference on genetic and evolutionary computation*. Morgan Kaufmann Publishers Inc., pp. 613–620.

M.C. Roberts, A.S. Dizier and J. Vaughan (2012). "Multiobjective optimization: portfolio optimization based on goal programming methods".

Microsoft (2019a). "Microsoft Surface Pro 7". URL: `https://www.microsoft.com/en-us/p/surface-pro-7/8n17j0m5zzqs`.

Microsoft (2019b). ".NET Framework 4.8". URL: `https://dotnet.microsoft.com/download/dotnet-framework/net48`.

Mohammadi, Mahmoud, Mahin Nastaran, and Alireza Sahebgharani (2016). "Development, application, and comparison of hybrid meta-heuristics for urban land-use allocation optimization: Tabu search, genetic, GRASP, and simulated annealing algorithms". *Computers, Environment and Urban Systems* 60, pp. 23–36.

Nebro, A. J. (2015). "jMetal.NET". URL: `http://jmetalnet.sourceforge.net/`.

Paquete, Luis, Marco Chiarandini, and Thomas Stützle (2004). "Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study". *Metaheuristics for multiobjective optimisation*. Springer, pp. 177–199.

Porta, Juan, Jorge Parapar, Ramon Doallo, Francisco F Rivera, Inés Santé, and Rafael Crecente (2013). "High performance genetic algorithm for land use planning". *Computers, Environment and Urban Systems* 37, pp. 45–58.

Santé-Riveira, Inés, Marcos Boullón-Magán, Rafael Crecente-Maseda, and David Miranda-Barrós (2008). "Algorithm based on simulated annealing for land-use allocation". *Computers & Geosciences* 34.3, pp. 259–268.

Schwaab, Jonas, Kalyanmoy Deb, Erik Goodman, Sven Lautenbach, Maarten J van Strien, and Adrienne Grêt-Regamey (2018). "Improving the performance of genetic algorithms for land-use allocation problems". *International Journal of Geographical Information Science* 32.5, pp. 907–930.

Sharma, Sunil Kumar (2005). "A comparison of combinatory methods and GIS based MOLA (IDRISI®) for solving multi-objective land use assessment and allocation problems".

Song, Mingjie and Dongmei Chen (2018). "An improved knowledge-informed NSGA-II for multi-objective land allocation (MOLA)". *Geo-spatial Information Science* 21.4, pp. 273–287.

Stewart, Theodor J, Ron Janssen, and Marjan van Herwijnen (2004). "A genetic algorithm approach to multiobjective land use planning". *Computers & Operations Research* 31.14, pp. 2293–2313.

Strauch, Michael, Anna F Cord, Carola Pätzold, Sven Lautenbach, Andrea Kaim, Christian Schweitzer, Ralf Seppelt, and Martin Volk (2019). "Constraints in multi-objective optimization of land use allocation–Repair or penalize?" *Environmental Modelling & Software* 118, pp. 241–251.

Verstegen, Judith A, Floor van der Hilst, Geert Woltjer, Derek Karssenberg, Steven M de Jong, and André PC Faaij (2016). "What can and can't we say about indirect land-use change in Brazil using an integrated economic–land-use change model?" *Gcb Bioenergy* 8.3, pp. 561–578.

Walter, Arnaldo, Marcelo Valadares Galdos, Fabio Vale Scarpare, Manoel Regis Lima Verde Leal, Joaquim Eugênio Abel Seabra, Marcelo Pereira da Cunha, Michelle Cristina Araujo Picoli, and Camila Ortolan Fernandes de Oliveira (2014). "Brazilian sugarcane ethanol: developments so far and challenges for the future". *Wiley Interdisciplinary Reviews: Energy and Environment* 3.1, pp. 70–92.

Whitley, Darrell (1994). "A genetic algorithm tutorial". *Statistics and computing* 4.2, pp. 65–85.

Wicke, Birka, Pita Verweij, Hans van Meijl, Detlef P van Vuuren, and Andre PC Faaij (2012). "Indirect land use change: review of existing models and strategies for mitigation". *Biofuels* 3.1, pp. 87–100.

Zitzler, E., K. Deb, L. Thiele, C. Coello, C. Coello, and D. Corne (2001). *Evolutionary Multi-criterion Optimization*. 1993. Springer Science & Business Media.

Appendix A

Metadata Initialization

A.1 MD-I

Algorithm 31 InitializeMD-I(s)

input: a solution s with:
    an array $f(s)$ for each $s \in \mathcal{A}$ with obj. values ▷ MD I

for all objective $o \in \mathcal{O}$ **do**
    $f(s)[o] := f_o(s)$
end for

output: s

Algorithm 32 UpdateMD-I($s, \mathcal{U}$)

input: a solution s with:
    an array $f(s)$ for each $s \in \mathcal{A}$ with obj. values ▷ MD I
    a set $\mathcal{U}$ containing the updates for s ▷ Definition 6.2.6

for all objective $o \in \mathcal{O}$ **do**
    $f(s)[o]$ += $f_o(s, \mathcal{U})$ ▷ Appendix A
end for

output: s

A.2 MD-III

Algorithm 33 InitializeMD-III(s)

input: a solution s with:
    an array $n(s)$ with the number of cells per type ▷ MD-III

for all land-use type $t \in \mathcal{T}$ **do**
    $n(s)[t] = 0$
end for

for all cell c in solution s **do**
    $n(s)[\text{type}(c)]$ += 1
end for

output: s

Algorithm 34 UpdateMD-III($s, \mathcal{U}$)

input: a solution s with:
        an array $n(s)$ with the number of cells per type ▷ MD-III
    a set $\mathcal{U}$ containing the updates for s ▷ Definition 6.2.6

for all cell $c \in \mathcal{U}$ **do**
    $n(s)[\text{type}(s[c])]$ -= 1
    $n(s)[\text{type}(c)]$ += 1
end for

output: s

Appendix B

Incremental Objective Functions

B.1 Development Costs

Algorithm 35 $f_{cost}(s, \mathcal{U})$

input: a solution s
        a set $\mathcal{U}$ containing the updates for s ▷ Definition 6.2.6

$\Delta f_{cost} = 0$
for all cell $c \in \mathcal{U}$ **do**
    $(x, y) := c$
    t_{old} := type(c) in s
    t_{new} := type(c) in $\mathcal{U}$
    Δf_{cost} -= $C_{xyt_{old}}$ ▷ Equation (3.9)
    Δf_{cost} += $C_{xyt_{new}}$
end for

output: Δf_{cost}

B.2 Compactness

Algorithm 36 $f_{comp}(s, \mathcal{U})$

input: a solution s
a set $\mathcal{U}$ containing the updates for s ▷ Definition 6.2.6

$s' := s$
for all cell $c \in \mathcal{C}$ **do**
  type($s'[c]$) := type(c)
end for

$\Delta f_{comp} = 0$
$\mathcal{C}' := \emptyset$
for all cell $c \in \mathcal{C}$ **do**
  for all cell $c' \in (\mathcal{N}(c) \cup c)$ **do**
    if $c' \notin \mathcal{C}'$ **then**
      $(x, y) := c'$
      Δf_{comp} -= $\sum_{m=x-1}^{x+1} \sum_{n=y-1}^{y+1} \frac{Neig_{mn}}{8}$ in s ▷ Equation (3.13)
      Δf_{comp} += $\sum_{m=x-1}^{x+1} \sum_{n=y-1}^{y+1} \frac{Neig_{mn}}{8}$ in s'
    end if
  end for
end for

output: Δf_{comp}

Appendix C

Incremental Constraint Validation

C.1 Allocation Ranges

Algorithm 37 Validate-I(s)

input: a solution s with the following metadata:
    an array $n(s)$ with the number of cells per type ▷ MD-III

$\mathcal{T}' := \{t \in \mathcal{T} \mid \text{!static}(t)\}$
for all land-use type $t \in \mathcal{T}$ **do**
    if $n(s)[t] < L_t$ or $n(s)[t] > U_t$ **then**
        output: $false$
    end if
end for

output: $true$

Appendix D

Optimized CD Calculation

D.1 CD-PLS

Algorithm 38 CD-PLS(s, $\mathcal{A}$)

input: a solution s
    an archive $\mathcal{A}$ with:
      an array $f(s)$ for each $s \in \mathcal{A}$ with obj. values ▷ MD I
      an array $v(\mathcal{A})$ with all $s \in \mathcal{A}$ sorted per obj. value ▷ MD II

$v' := v(\mathcal{A})$
if s is not included in $v(\mathcal{A})$ **then**
    v' := insert s in $v'[o]$ for all objectives $o \in \mathcal{O}$ using array $f(s)$
end if

$l := |\mathcal{A}|$
$cd := 0$
for all objective $o \in \mathcal{O}$ **do**
    i := position of s in $v'[o]$
    $s_{min} := v'[o][0]$
    $s_{max} := v'[o][l]$
    if $s == s_{min}$ **then**
        $s_{suc} := v'[o][1]$
        cd += $\frac{2(f_o(s_{suc}) - f_o(s))}{f_o(s_{max}) - f_o(s_{min})}$
    else if $s == s_{max}$ **then**
        $s_{prec} := v'[o][l-1]$
        cd += $\frac{2(f_o(s) - f_o(s_{prec}))}{f_o(s_{max}) - f_o(s_{min})}$
    else
        $s_{prec} := v'[o][i-1]$
        $s_{suc} := v'[o][i+1]$
        cd += $\frac{f_o(s_{suc}) - f_o(s_{prec})}{f_o(s_{max}) - f_o(s_{min})}$
    end if
end for

output: cd

Appendix E

Implementation Structures

E.1 Other Framework Structures

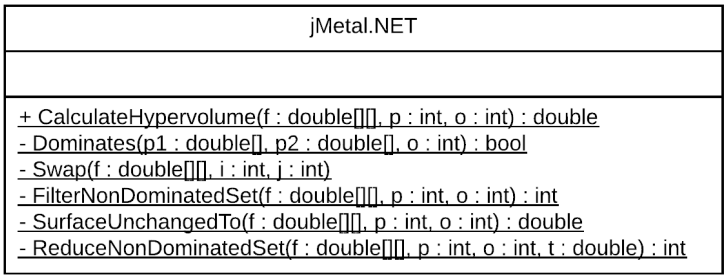


FIGURE E.1: UML Class: jMetal.NET

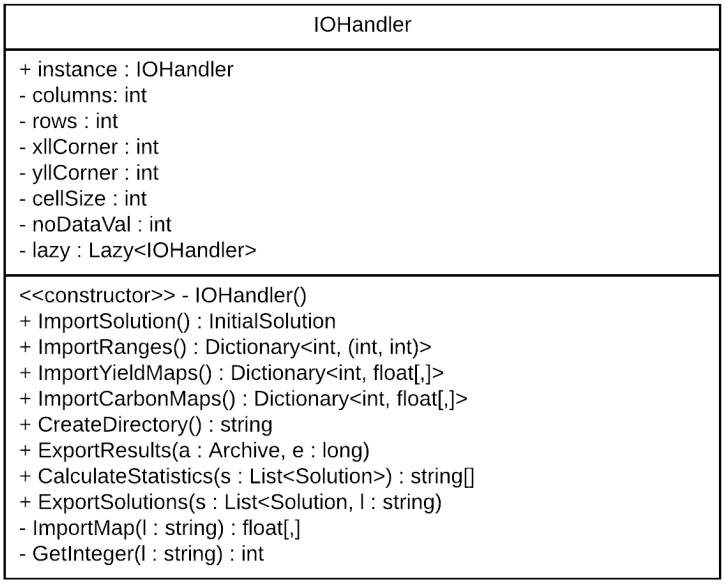


FIGURE E.2: UML Class: IOHandler

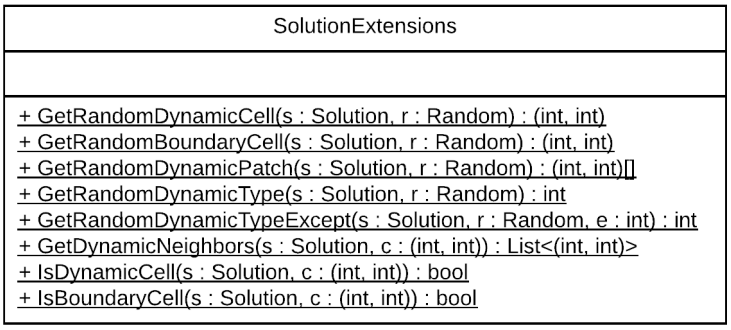


FIGURE E.3: UML Class: SolutionExtensions

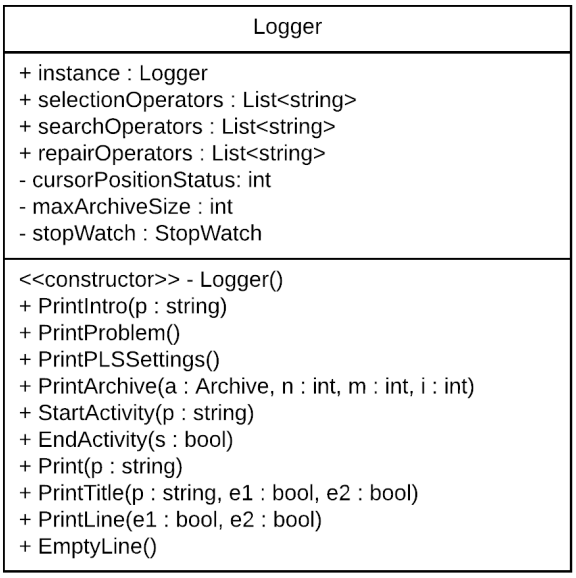


FIGURE E.4: UML Class: Logger

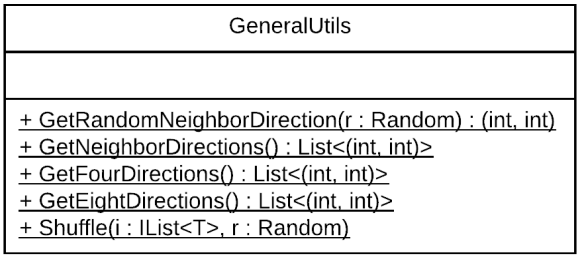


FIGURE E.5: UML Class: GeneralUtils

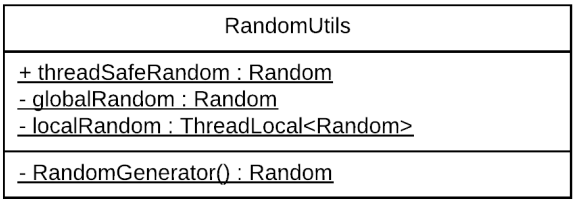


FIGURE E.6: UML CLass: RandomUtils

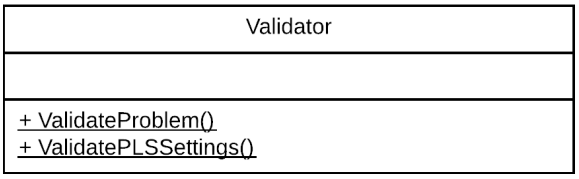


FIGURE E.7: UML Class: Validator

E.2 NSGA-II Framework Extensions

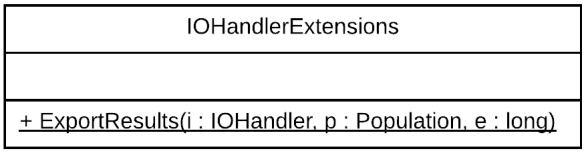


FIGURE E.8: UML Class: IOHandlerExtensions

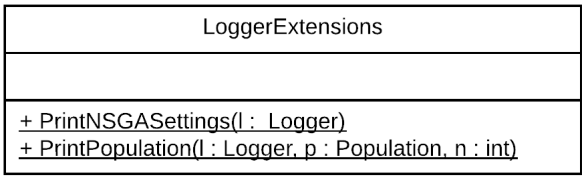


FIGURE E.9: UML Class: LoggerExtensions

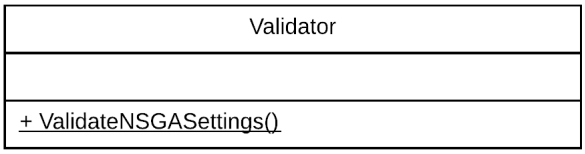


FIGURE E.10: UML Class: Validator

Appendix F

Implementation Statistics

F.1 Framework and (I)PLS-MOLA

TABLE F.1: Implementation Statistics MOLA Framework, PLS-MOLA and IPLS-MOLA

Class	**Maintainability Index**	**Cyclomatic Complexity**	**Depth of Inheritance**	**Class Coupling**	**Lines of Exec. Code**
Program	57	1	1	10	16
IPLS	49	16	1	36	45
PLS	54	20	1	15	48
Archive	62	37	1	16	80
DerivedSolution	81	15	2	6	21
InitialSolution	86	15	2	13	16
Solution	88	25	1	9	17
GeneralUtils	79	7	1	5	11
jMetalNET	63	22	1	3	48
RandomUtils	84	3	1	3	7
SolutionExtensions	65	31	1	6	51
Validator	53	37	1	4	25
IOHandler	55	51	1	40	176
Logger	65	37	1	22	87
Constraint	100	3	1	1	0
Objective	100	4	1	3	0
Problem	73	10	1	16	24
AllocationRanges	80	13	2	6	12
Carbon	74	8	2	6	10
Compactness	68	10	2	8	17
Yield	74	8	2	6	10
Settings	76	75	3	7	152
RepairOperator	91	4	1	5	3
RepairProcedure	69	8	1	7	10
LR_BRBM	52	17	2	16	34
LR_BRCM	54	14	2	15	30
SearchOperator	96	4	1	3	1
SearchProcedure	90	2	1	4	2
LS_KRBM	60	10	2	11	20
LS_KRCM	60	7	2	11	21
LS_KRPM	59	10	2	10	20
SelectionOperator	100	3	1	2	0
SelectionProcedure	81	2	1	7	4
S_CD	74	4	2	4	9
S_R	90	2	2	4	2
Total:	72	535	3	112	1029

F.2 NSGA-II-MOLA

TABLE F.2: Implementation Statistics MOLA Framework Extensions and NSGA-II-MOLA

Class	**Maintainability Index**	**Cyclomatic Complexity**	**Depth of Inheritance**	**Class Coupling**	**Lines of Executable code**
Program	44	1	1	15	35
NSGA_II	43	18	1	41	69
CrossoverOperator	100	3	1	3	0
CrossoverProcedure	89	2	1	5	2
C_XBC	63	10	2	12	17
C_XTD	66	8	2	12	10
Population	61	54	1	23	108
Validator	51	24	1	2	14
InitializationProcedure	54	26	1	31	72
IOHandlerExtensions	41	5	1	26	39
LoggerExtensions	48	13	1	18	42
MutationProcedure	87	3	1	5	3
Settings	76	69	3	7	140
TournamentProcedure	53	8	1	16	16
Total	62	244	3	114	567

Appendix G

Run Results

G.1 Run 1 - 6

TABLE G.1: Running Times (ms) of Run 1 - 6

Run	**1**	**2**	**3**	**4**	**5**	**6**
	612818	541494	552159	522693	520271	341399
	658485	612181	586219	532129	542986	447047
	718856	618968	605981	549064	545162	429192
	693430	606166	539557	626541	598672	404151
	628616	555639	539380	594669	456044	412843
$\bar{x}$	**662441.0**	**586889.6**	**564659.2**	**565019.2**	**532627.0**	**406926.4**
s	44060.17	35628.71	34107.74	44135.82	51584.89	40130.94

G.2 Run 7 - 9

TABLE G.2: Hypervolumes ($\times 10^{14}$) of Run 7 - 9

Run	**7**	**8**	**9**
	3.67628	3.70711	3.69615
	3.69518	3.70287	3.70080
	3.69067	3.70000	3.69000
	3.70496	3.70391	3.68679
	3.69882	3.70131	3.70013
$\bar{x}$	**3.69318**	**3.70380**	**3.69550**
s	0.01080	0.00212	0.00569

G.3 Run 10 - 12

TABLE G.3: Running Times (ms) of Run 10 - 12

Run	**10**	**11**	**12**
	460796	558406	554747
	534483	589476	547417
	493112	530039	538714
	525245	548134	539442
	516490	581331	541976
$\bar{x}$	**506025.2**	**561477.2**	**544459.2**
s	29585.43	24258.71	6683.298

G.4 Run 13 - 19

TABLE G.4: Hypervolumes ($\times 10^{14}$) of Run 13 - 19

Run	13	14	15	16	17	18	19
	3.66453	3.69898	3.68220	3.67555	3.68887	3.68463	3.69517
	3.67414	3.69135	3.68060	3.68703	3.68027	3.69549	3.68932
	3.67091	3.69342	3.68560	3.68147	3.68270	3.68607	3.69265
	3.67232	3.70131	3.67386	3.67566	3.68502	3.69098	3.68072
	3.67206	3.69749	3.68007	3.67440	3.68457	3.68287	3.69689
$\bar{x}$	**3.67079**	**3.69651**	**3.68046**	**3.67882**	**3.68429**	**3.68801**	**3.69095**
s	0.00369	0.00407	0.00428	0.00535	0.00317	0.00516	0.00639

G.5 Run 20 - 26

TABLE G.5: Running Times (ms) of Run 20 - 26

Run	20	21	22	23	24	25	26
	263369	148592	1217627	226263	805715	731474	560349
	249341	139496	1262131	226574	843405	724253	627410
	233406	162518	1241599	212111	813393	722654	609634
	234753	173797	1231391	277736	874342	712305	575992
	247439	178492	1256876	268304	857270	713691	568142
$\bar{x}$	**245661.6**	**160579.0**	**1241924.8**	**242197.6**	**838825.0**	**720875.4**	**588305.4**
s	12241.80	16488.87	18257.047	28929.80	29007.37	7936.770	28826.30

G.6 Run 27 - 34

TABLE G.6: Hypervolumes ($\times 10^{14}$) of Run 27 - 34

Run	27	28	29	30	31	32	33	34
	3.69700	3.69817	3.68848	3.68567	3.68924	3.69389	3.68651	3.69409
	3.68784	3.69173	3.68752	3.68584	3.68699	3.69197	3.69085	3.68571
	3.68558	3.68816	3.69706	3.68351	3.68852	3.68481	3.68589	3.69000
	3.69739	3.68309	3.68393	3.68923	3.69316	3.68043	3.69464	3.69298
	3.69010	3.68731	3.67844	3.68910	3.68794	3.69530	3.69110	3.70000
$\bar{x}$	**3.69158**	**3.68969**	**3.68709**	**3.68667**	**3.68917**	**3.68928**	**3.68980**	**3.69256**
s	0.00537	0.00565	0.00683	0.00246	0.00238	0.00639	0.00362	0.00528

G.7 Run 35 - 41

TABLE G.7: Hypervolumes ($\times 10^{14}$) of Run 35 - 41

Run	35	36	37	38	39	40	41
	3.72840	3.71972	3.72212	3.68567	3.67059	3.66194	3.66187
	3.75186	3.73230	3.70819	3.68584	3.66986	3.66370	3.65749
	3.74185	3.73757	3.71642	3.68351	3.67125	3.66555	3.66107
	3.74149	3.73066	3.71856	3.68923	3.66787	3.66171	3.65959
	3.74360	3.73616	3.70604	3.68910	3.67006	3.67159	3.65663
$\bar{x}$	**3.74144**	**3.73128**	**3.71427**	**3.68667**	**3.66993**	**3.66490**	**3.65933**
s	0.00842	0.00704	0.00688	0.00246	0.00127	0.00405	0.00225

G.8 Run 42 - 44

TABLE G.8: Hypervolumes ($\times 10^{14}$) of Run 42 - 44

Run	42	43	44
	3.65037	3.70398	3.70169
	3.64676	3.70038	3.68617
	3.64862	3.69550	3.69821
	3.65461	3.69138	3.69977
	3.64992	3.70093	3.69511
$\bar{x}$	**3.65006**	**3.69843**	**3.69619**
s	0.00291	0.00498	0.00610

G.9 Run 45 - 47

TABLE G.9: Running Times (ms) of Run 45 - 47

Run	45	46	47
	319935	909833	717052
	355946	960027	715959
	377232	958610	714303
	392560	985532	709244
	288681	958997	716486
$\bar{x}$	**346870.8**	**954599.8**	**714608.8**
s	42436.00	27503.36	3169.808

G.10 Run 48 - 53

TABLE G.10: Hypervolumes ($\times 10^{14}$) of Run 48 - 53

Run	48	49	50	51	52	53
	3.42550	3.64936	3.72998	3.74577	3.75611	3.75788
	3.42318	3.64736	3.72932	3.74386	3.75518	3.75937
	3.42313	3.64743	3.72760	3.74387	3.75775	3.76014
	3.41928	3.64814	3.73193	3.74265	3.75704	3.75818
	3.42301	3.64877	3.72684	3.74371	3.75459	3.76034
$\bar{x}$	**3.42282**	**3.64821**	**3.72913**	**3.74397**	**3.75613**	**3.75918**
s	0.00224	0.00086	0.00201	0.00113	0.00130	0.00111

TABLE G.11: Running Times (ms) of Run 48 - 53

Run	48	49	50	51	52	53
	38738	48673	106542	174526	389476	779389
	39172	49868	102794	174040	385934	770601
	41865	48394	103263	172481	375868	778981
	41172	46516	103809	175024	383598	776255
	41131	46736	103086	177376	382478	773838
$\bar{x}$	**40415.6**	**48037.4**	**103898.8**	**174689.4**	**383470.8**	**775812.8**
s	1373.46	1404.51	1523.158	1778.709	5023.300	3677.123

G.11 Run 54 - 55

TABLE G.12: Hypervolumes ($\times 10^{14}$) of Run 54 - 55

Run	54	55
	3.62052	3.68638
	3.62803	3.68621
	3.62268	3.68967
	3.62993	3.68656
	3.62183	3.68070
$\bar{x}$	**3.62460**	**3.68590**
s	0.00413	0.00324

G.12 Run 56 - 57

TABLE G.13: Hypervolumes ($\times 10^{14}$) of Run 56 - 57

Run	56	57
	3.67149	3.69020
	3.67661	3.69106
	3.67750	3.68697
	3.67435	3.69327
	3.67200	3.67716
$\bar{x}$	**3.67439**	**3.68773**
s	0.00268	0.00633

G.13 Run 58 - 59

TABLE G.14: Hypervolumes ($\times 10^{14}$) of Run 56 - 57

Run	58	59
	3.68997	3.72251
	3.68855	3.71682
	3.68791	3.72038
	3.68586	3.71652
	3.68771	3.71714
$\bar{x}$	**3.68800**	**3.71867**
s	0.00149	0.00265

TABLE G.15: Archive Size of Run 58 - 59

Run	58	59
	12	6
	13	2
	22	3
	8	3
	34	6
$\bar{x}$	**17.8**	**4.0**
s	10.4	1.9

G.14 Run 60 - 66

TABLE G.16: Hypervolumes ($\times 10^{14}$) of Run 60 - 66

Run	60	61	62	63	64	65	66
	3.69349	3.68469	3.68954	3.69706	3.68105	3.69221	3.69606
	3.68622	3.68177	3.69454	3.68584	3.69664	3.69370	3.68703
	3.68558	3.68431	3.68354	3.69415	3.68850	3.68566	3.69946
	3.68149	3.67208	3.68838	3.68595	3.68540	3.69047	3.69587
	3.68887	3.68147	3.67580	3.68850	3.67706	3.68541	3.68863
$\bar{x}$	**3.68713**	**3.68086**	**3.68636**	**3.69030**	**3.68573**	**3.68949**	**3.69341**
s	0.00443	0.00640	0.00635	0.00506	0.00748	0.00379	0.00532

G.15 Run 67 - 68

TABLE G.17: Running Times (ms) of Run 67 - 68

Run	67	68
	840222	433504
	826591	397779
	883274	407299
	785946	431945
	844288	457952
$\bar{x}$	**836064.2**	**425695.8**
s	35054.45	23760.64

TABLE G.18: Archive Size of Run 67 - 68

Run	67	68
	12	1029
	11	802
	11	1105
	18	1371
	23	1370
$\bar{x}$	**15.0**	**1135.4**
s	5.34	241.84

G.16 Run 69 - 70

TABLE G.19: Hypervolumes ($\times 10^{14}$) of Run 69 - 70

Run	69	70
	3.68825	3.57297
	3.68692	3.56567
	3.69048	3.57500
	3.69413	3.57451
	3.69277	3.57138
$\bar{x}$	**3.69051**	**3.57191**
s	0.00301	0.00377

Appendix H

Performance Results

H.1 PLS

TABLE H.1: Hypervolumes ($\times 10^{14}$) of PLS-MOLA for Brazil

Min	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
0	3.74277	3.74301	3.74297	3.74463	3.73978	**3.74263**	0.00176
15	3.88728	3.92236	3.90136	3.93725	3.89635	**3.90892**	0.02041
30	3.98476	4.03549	4.01359	4.02978	4.00017	**4.01276**	0.02091
45	4.05618	4.09208	4.07458	4.09635	4.06857	**4.07755**	0.01666
60	4.12195	4.14342	4.13562	4.14625	4.13745	**4.13694**	0.00943
75	4.16424	4.18114	4.17024	4.18726	4.17152	**4.17488**	0.00920
90	4.19452	4.21194	4.20145	4.21726	4.20736	**4.20651**	0.00888
105	4.22151	4.23267	4.22847	4.23645	4.23023	**4.22987**	0.00555
120	4.24463	4.25007	4.24857	4.25734	4.25362	**4.25085**	0.00486
135	4.26412	4.27111	4.27002	4.27827	4.28015	**4.27273**	0.00652
150	4.27912	4.28925	4.28445	4.29014	4.29327	**4.28725**	0.00553

TABLE H.2: Results of PLS-MOLA for Brazil

Result	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
Max. Comp. ($\times 10^5$)	6.9487	6.9862	7.0056	7.0164	6.9460	**6.9806**	0.0322
Avg. Comp. ($\times 10^5$)	6.9486	6.9643	6.9875	7.0032	6.9458	**6.9699**	0.0249
Min. Comp. ($\times 10^5$)	6.9485	6.9424	6.9684	6.9873	6.9456	**6.9584**	0.0191
Max. Yield ($\times 10^8$)	6.1478	6.1790	6.1839	6.1859	6.1751	**6.1744**	0.0154
Avg. Yield ($\times 10^8$)	6.1475	6.1784	6.1834	6.1854	6.1747	**6.1739**	0.0153
Min. Yield ($\times 10^8$)	6.1472	6.1780	6.1830	6.1850	6.1743	**6.1735**	0.0153

TABLE H.3: Hypervolumes ($\times 10^{13}$) of PLS-MOLA for Centre West

Min	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
0	2.03672	2.03185	2.03831	2.03425	2.03655	**2.03554**	0.00252
15	2.14253	2.13447	2.13057	2.12734	2.14123	**2.13523**	0.00659
30	2.17492	2.17325	2.17342	2.16659	2.17342	**2.17232**	0.00327
45	2.19530	2.19789	2.20302	2.19802	2.19230	**2.19731**	0.00395
60	2.22666	2.21354	2.22596	2.21574	2.23666	**2.22371**	0.00933
75	2.23982	2.23014	2.24505	2.24213	2.24006	**2.23944**	0.00561
90	2.24596	2.24358	2.25203	2.25005	2.24674	**2.24767**	0.00336
105	2.24978	2.25826	2.25712	2.25622	2.25538	**2.25535**	0.00329
120	2.25662	2.26706	2.26112	2.26202	2.26002	**2.26137**	0.00378
135	2.26342	2.27746	2.26971	2.26641	2.26442	**2.26828**	0.00566
150	2.27242	2.28626	2.28012	2.27151	2.27443	**2.27695**	0.00619

TABLE H.4: Results of PLS-MOLA for Centre West

Result	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
Max. Comp. ($\times 10^5$)	1.2520	1.2525	1.2503	1.2519	1.2542	**1.2522**	0.0016
Avg. Comp. ($\times 10^5$)	1.2520	1.2520	1.2501	1.2519	1.2541	**1.2520**	0.0017
Min. Comp. ($\times 10^5$)	1.2519	1.2518	1.2498	1.2519	1.2540	**1.2519**	0.0017
Max. Yield ($\times 10^8$)	1.8220	1.8484	1.8130	1.8200	1.8454	**1.8317**	0.0179
Avg. Yield ($\times 10^8$)	1.8220	1.8434	1.8124	1.8200	1.8453	**1.8303**	0.0166
Min. Yield ($\times 10^8$)	1.8218	1.8432	1.8121	1.8200	1.8451	**1.8301**	0.0165

TABLE H.5: Hypervolumes ($\times 10^{11}$) of PLS-MOLA for Sul Goiano

Min	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
0	0.87892	0.88970	0.88542	0.87467	0.88438	**0.88262**	0.00587
15	1.12045	1.11755	1.13005	1.11952	1.12641	**1.12280**	0.00523
30	1.14871	1.15689	1.15539	1.14812	1.15593	**1.15301**	0.00423
45	1.17549	1.17505	1.17609	1.17294	1.17849	**1.17561**	0.00200
60	-	1.18921	1.18801	-	-	**1.18861**	0.08453
75	-	-	-	-	-	-	-

TABLE H.6: Results of PLS-MOLA for Sul Goiano

Result	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
Max. Comp. ($\times 10^3$)	7.7368	7.8108	7.7798	7.8738	7.8363	**7.8075**	0.0524
Avg. Comp. ($\times 10^3$)	7.7170	7.8078	7.7721	7.8703	7.8323	**7.7999**	0.0585
Min. Comp. ($\times 10^3$)	7.6978	7.8058	7.7618	7.8638	7.8248	**7.7908**	0.0636
Max. Yield ($\times 10^7$)	1.4300	1.4400	1.4289	1.4762	1.4633	**1.4477**	0.0211
Avg. Yield ($\times 10^7$)	1.4300	1.4400	1.4282	1.4756	1.4625	**1.4472**	0.0209
Min. Yield ($\times 10^7$)	1.4200	1.4400	1.4269	1.4746	1.4615	**1.4446**	0.0230

H.2 IPLS

TABLE H.7: Hypervolumes ($\times 10^{11}$) of IPLS-MOLA for Sul Goiano with Perturbation Size = 25

Min	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
0	0.88513	0.88894	0.88754	0.87043	0.88936	**0.88428**	0.00792
30	1.16152	1.15524	1.15754	1.16017	1.16156	**1.15921**	0.00275
60	1.18278	1.19050	1.19204	1.18954	1.18834	**1.18864**	0.00354
90	1.20540	1.20667	1.21326	1.20843	1.21738	**1.21023**	0.00499
120	1.22769	1.20954	1.22946	1.21537	1.22269	**1.22095**	0.00840
150	1.23114	1.21792	1.23962	1.22683	1.22986	**1.22907**	0.00785

TABLE H.8: Hypervolumes ($\times 10^{11}$) of IPLS-MOLA for Sul Goiano with Perturbation Size = 50

Min	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
0	0.887052	0.882119	0.885246	0.884233	0.886714	**0.88507**	0.00200
30	1.13030	1.13912	1.13542	1.13245	1.08653	**1.12476**	0.02163
60	1.18649	1.19493	1.18532	1.18367	1.19608	**1.18930**	0.00577
90	1.20649	1.21237	1.20489	1.20730	1.21459	**1.20913**	0.00414
120	1.22470	1.21251	1.20850	1.21652	1.20067	**1.21258**	0.00896
150	1.22470	1.22164	1.21542	1.21862	1.21206	**1.21849**	0.00498

TABLE H.9: Hypervolumes ($\times 10^{11}$) of IPLS-MOLA for Sul Goiano with Perturbation Size = 75

Min	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
0	0.88209	0.88314	0.88256	0.88164	0.88433	**0.88275**	0.00104
30	1.12855	1.12278	1.13067	1.12432	1.12264	**1.12579**	0.00363
60	1.20182	1.20033	1.20396	1.19543	1.20023	**1.20035**	0.00314
90	1.20413	1.20137	1.21515	1.19948	1.20406	**1.20484**	0.00609
120	1.21720	1.21308	1.22057	1.20679	1.20969	**1.21347**	0.00555
150	1.21720	1.21311	1.23013	1.20743	1.20969	**1.21551**	0.00897

TABLE H.10: Results of IPLS-MOLA for Sul Goiano with Perturbation Size = 25

Result	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
Max. Comp. ($\times 10^3$)	7.7828	7.7388	7.7885	7.7686	7.7813	**7.7720**	0.0199
Avg. Comp. ($\times 10^3$)	7.7466	7.7134	7.7528	7.7355	7.7415	**7.7380**	0.0151
Min. Comp. ($\times 10^3$)	7.6683	7.6953	7.6724	7.6743	7.6588	**7.6738**	0.0134
Max. Yield ($\times 10^7$)	1.5800	1.5700	1.5880	1.5770	1.5810	**1.5792**	0.0065
Avg. Yield ($\times 10^7$)	1.5800	1.5400	1.5840	1.5650	1.5740	**1.5686**	0.0175
Min. Yield ($\times 10^7$)	1.5800	1.4100	1.5800	1.5430	1.5680	**1.5362**	0.0721

H.3 NSGA-II

TABLE H.11: Hypervolumes ($\times 10^{14}$) of NSGA-II-MOLA for Brazil

Min	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
0	2.93041	2.93386	2.94021	2.93862	2.92872	**2.93436**	0.00500
30	3.05841	3.03664	3.04894	3.04321	3.03325	**3.04409**	0.01003
60	3.06146	3.03881	3.07081	3.06726	3.03561	**3.05479**	0.01643
90	3.07554	3.06346	3.08636	3.07234	3.06026	**3.07159**	0.10354
120	3.07648	3.07274	3.09031	3.07644	3.06857	**3.07691**	0.00817
150	3.09147	3.07846	3.10064	3.08236	3.07428	**3.08544**	0.01061

TABLE H.12: Results of NSGA-II-MOLA for Brazil

Result	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
Max. Comp. ($\times 10^5$)	5.7619	5.7620	5.7654	5.7603	5.7590	**5.7618**	0.0024
Avg. Comp. ($\times 10^5$)	5.7021	5.6985	5.7062	5.7003	5.6937	**5.7001**	0.0046
Min. Comp. ($\times 10^5$)	5.5072	5.5608	5.5096	5.5692	5.5284	**5.5351**	0.0287
Max. Yield ($\times 10^8$)	5.3696	5.3473	5.3705	5.3582	5.3385	**5.3568**	0.0139
Avg. Yield ($\times 10^8$)	5.1743	5.1704	5.1768	5.1724	5.1689	**5.1726**	0.0031
Min. Yield ($\times 10^8$)	5.0837	5.0865	5.0857	5.0858	5.0832	**5.0850**	0.0014

TABLE H.13: Hypervolumes ($\times 10^{13}$) of NSGA-II-MOLA for Centre West

Min	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
0	1.48025	1.47751	1.46861	1.48641	1.46891	**1.47634**	0.00763
30	1.55192	1.53855	1.53485	1.54035	1.52678	**1.53849**	0.00914
60	1.60356	1.57429	1.56939	1.57666	1.57234	**1.57925**	0.01385
90	1.63777	1.61111	1.60453	1.62471	1.62234	**1.62009**	0.01286
120	1.66623	1.63644	1.62565	1.64832	1.63789	**1.64291**	0.01531
150	1.69605	1.65941	1.63831	1.67022	1.64932	**1.66266**	0.02210

TABLE H.14: Results of NSGA-II-MOLA for Centre West

Result	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
Max. Comp. ($\times 10^5$)	1.0094	1.0038	0.9973	1.0033	0.9992	**1.0026**	0.0047
Avg. Comp. ($\times 10^5$)	1.0020	0.9965	0.9927	0.9972	0.9939	**0.9964**	0.0036
Min. Comp. ($\times 10^5$)	0.9905	0.9848	0.9804	0.9884	0.9823	**0.9854**	0.0042
Max. Yield ($\times 10^8$)	1.6808	1.6537	1.6329	1.6714	1.6418	**1.6561**	0.0200
Avg. Yield ($\times 10^8$)	1.6425	1.6154	1.5967	1.6314	1.6003	**1.6173**	0.0197
Min. Yield ($\times 10^8$)	1.5944	1.5718	1.5439	1.5862	1.5646	**1.5722**	0.0197

TABLE H.15: Hypervolumes ($\times 10^{11}$) of NSGA-II-MOLA for Sul Goiano

Min	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
0	0.61271	0.61337	0.61186	0.61353	0.61294	**0.61288**	0.00066
30	0.97119	0.97386	0.96897	0.97604	0.97235	**0.97248**	0.00267
60	1.00737	1.02092	0.99899	1.02874	0.99985	**1.01117**	0.01318
90	1.02895	1.03607	1.02634	1.04017	1.02454	**1.03121**	0.00666
120	1.05469	1.05697	1.05348	1.05863	1.05323	**1.05540**	0.00233
150	1.06485	1.06619	1.05942	1.06736	1.06384	**1.06433**	0.00305

TABLE H.16: Results of NSGA-II-MOLA for Sul Goiano

Result	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
Max. Comp. ($\times 10^3$)	7.0963	7.1503	7.0738	7.1633	7.0950	**7.1157**	0.0388
Avg. Comp. ($\times 10^3$)	7.0324	7.0912	7.0194	7.1042	7.0275	**7.0549**	0.0396
Min. Comp. ($\times 10^3$)	6.9633	7.0053	6.9633	7.0118	6.9428	**6.9773**	0.0298
Max. Yield ($\times 10^7$)	1.5000	1.4900	1.4810	1.4960	1.4920	**1.4918**	0.0072
Avg. Yield ($\times 10^7$)	1.4900	1.4800	1.4750	1.4910	1.4800	**1.4832**	0.0070
Min. Yield ($\times 10^7$)	1.4700	1.4700	1.4730	1.4750	1.4680	**1.4712**	0.0028

TABLE H.17: Hypervolumes ($\times 10^{11}$) of NSGA-II-MOLA for Sul Goiano with Initial Map in Initial Population

Min	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
0	0.64723	0.64992	0.65068	0.64723	0.65164	**0.64934**	0.00202
30	0.97170	0.98818	0.99115	0.96732	0.99564	**0.98280**	0.01251
60	1.01529	1.02553	1.02743	1.01276	1.03524	**1.02325**	0.00922
90	1.03790	1.04292	1.04375	1.03562	1.04625	**1.04129**	0.00439
120	1.05527	1.05632	1.05714	1.05496	1.05874	**1.05649**	0.00153
150	1.06265	1.0661	1.06635	1.05993	1.06764	**1.06454**	0.00317

TABLE H.18: Results of NSGA-II-MOLA for Sul Goiano with Initial Map in Initial Population

Result	Run 1	Run 2	Run 3	Run 4	Run 5	$\bar{x}$	s
Max. Comp. ($\times 10^3$)	7.0873	7.1053	7.1498	7.0773	7.1595	**7.1158**	0.0370
Avg. Comp. ($\times 10^3$)	7.0246	7.0439	7.0862	7.0235	7.1058	**7.0568**	0.0373
Min. Comp. ($\times 10^3$)	6.9223	6.9503	7.0095	6.9718	7.0143	**6.9736**	0.0391
Max. Yield ($\times 10^7$)	1.5010	1.4987	1.4960	1.4830	1.4940	**1.4945**	0.0070
Avg. Yield ($\times 10^7$)	1.4900	1.4908	1.4830	1.4710	1.4850	**1.4840**	0.0080
Min. Yield ($\times 10^7$)	1.4680	1.4780	1.4740	1.4750	1.4775	**1.4745**	0.0040

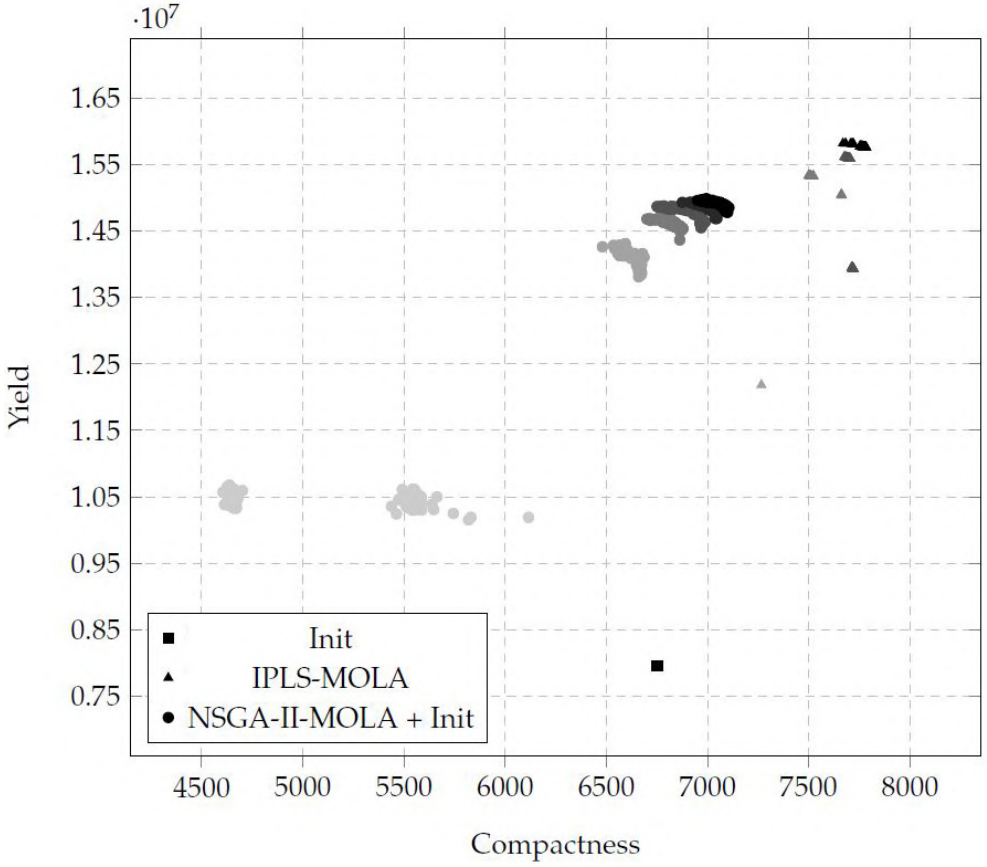


FIGURE H.1: Convergence of IPLS-MOLA and NSGA-II-MOLA with the Initial Map of Sul Goiano in the Initial Population