Utrecht University

Master Thesis in Artificial Intelligence

GOMEA-SAT

Applying Gene-pool Optimal Mixing Evolutionary Algorithm for the Boolean Satisfiability Problem

Author:
Krzysztof Leszek Sadowski, BSc

Student Number:
3611914

Supervisor:
Dr. ir. Dirk Thierens

July 10, 2012

Abstract

This paper explains the process of creating and optimizing GOMEA-SAT. It is a new Genetic Algorithm designed to solve the Satisfiability Problem in a fashion which is competitive with currently existing stochastic search solvers. A closer look is taken into the Gene-pool Optimal Mixing Evolutionary Algorithm. This algorithm is adapted to solve the SAT Problem, then modified and extensively tested in order to acquire the most optimal results. A local search is then added into the GOMEA-SAT and the results are contrasted against known SAT Problem solving algorithms such as Walksat and GASAT. Finally, Linkage Tree GA is modified and used to determine if learning the structure of a SAT Problem could be a next step in improving its solutions.

Contents

1 Introduction **4**
- 1.1 Problem Description . . . 4
- 1.2 Boolean Satisfiability . . . 4
  - 1.2.1 SAT Problem . . . 4
  - 1.2.2 MAX-SAT Problem . . . 5
  - 1.2.3 Motivation . . . 7
- 1.3 Genetic Algorithm . . . 7
- 1.4 GASAT . . . 8
  - 1.4.1 Selection . . . 8
  - 1.4.2 Recombination . . . 9
  - 1.4.3 Local Search . . . 10
  - 1.4.4 Insertion . . . 11
- 1.5 Walksat . . . 12

2 GOMEA **13**
- 2.1 LTGA . . . 13
- 2.2 GOMEA-SAT . . . 15
- 2.3 Clause to FOS Mapping . . . 16

3 GOMEA-SAT Experimental Study **18**
- 3.1 Study Overview . . . 18
- 3.2 Optimization Techniques and Results . . . 19
  - 3.2.1 Population vs. Evolution Length . . . 19
  - 3.2.2 Random FOS . . . 21
  - 3.2.3 FOS Adjustments (Singletons) . . . 22
  - 3.2.4 Tournament Selection . . . 24
  - 3.2.5 Forced Improvement . . . 25
  - 3.2.6 Conclusions . . . 27

4 Local Search **29**
- 4.1 Local Search Integration . . . 29
- 4.2 Walksat, Multistart Walksat and GASAT . . . 30
- 4.3 Small LS Results . . . 31
- 4.4 Medium LS Results . . . 32
- 4.5 Large LS Results . . . 32
- 4.6 GASAT Comparison . . . 33

4.7 Results Conclusion 34

5 Learning **37**
5.1 Overview 37
5.2 Configuration 38
5.3 Results 39

6 Conclusions **41**
6.1 Summary 41
6.2 Future Work 42

A Benchmark Overview **43**

Chapter 1

Introduction

1.1 Problem Description

Research Goal: *Applying Gene-pool Optimal Mixing Evolutionary techniques enhanced with a local search algorithm for the Boolean Satisfiability Problem will result in a new SAT-solving method (GOMEA-SAT) which will match the performances of other currently existing algorithms and due to its learning capacity often produce significantly better results.*

Boolean Satisfiability Problem (SAT Problem) is a very well known problem in the science community, with many real-world as well as academic applications. Due to its NP-complete nature many approaches have been created in order to generate solutions as optimal as possible. This thesis focuses on applying evolutionary techniques into the SAT Problem, by exploiting the existing structure of the SAT Problem with the use of the Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) and some variations of it, and in turn creating a new algorithm GOMEA-SAT. This optimized technique is then combined with a Walksat local search, in order to further improve the results. Those results are empirically tested and compared over a series of benchmarks against other existing SAT Problem solving algorithms such as GASAT, stand-alone Walksat, and Linkage Tree Learning Algorithm. All this methodologies will be explained in the following sections.

1.2 Boolean Satisfiability

1.2.1 SAT Problem

Boolean Satisfiability Problem (SAT Problem) is considered to be one of the best known NP-complete problems. It is a widely used modeling framework for solving combinatorial problems [3]. A SAT Problem instance consists of a set of Boolean variables:

$$X = \{x_1, x_2, x_3, .., x_n\}$$

as well as a Boolean formula

$$F : \{0, 1\}^n \rightarrow \{0, 1\}.$$

The goal in solving an instance of a Satisfiability Problem is to determine if variables of the given Boolean formula can be assigned in such a way, that with the given variable assignments, the entire formula evaluates to TRUE.

The Boolean formula for the SAT Problem is a logical conjunction of a set of clauses

$$C = \{c_1, c_2, c_3, .., c_m\}$$

resulting in the Boolean formula being of form

$$c_1 \wedge c_2 \wedge c_3 \wedge, .., \wedge c_m$$

Each clause (c_i) in turn is a logical disjunction of a subset of the Boolean variables present in a given SAT Problem. The idea behind the structure of a SAT Problem can be easier explained with the following simple example.

Let's look at a problem with only four Boolean variables $X = \{p_1, p_2, p_3, p_4\}$ and a set of four clauses [4]:

$$(p_1 \vee p_2 \vee \neg p_3), (\neg p_1 \vee p_2 \vee p_3), (\neg p_1 \vee \neg p_2 \vee p_3), (p_1 \vee \neg p_3 \vee p_4)$$

With the earlier definition, this gives us the Boolean formula for this example instance of the SAT Problem:

$$(p_1 \vee p_2 \vee \neg p_3) \wedge (\neg p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_2 \vee p_3) \wedge (p_1 \vee \neg p_3 \vee p_4)$$

The goal in solving a SAT problem is to determine if variables of a Boolean formula can be assigned in a way which will cause the formula to evaluate to TRUE. This means finding an assignment $v : X \to \{0, 1\}$, which satisfies the formula, if such assignment exists. In this example such result can be achieved with the following assignment [4]:

$$\begin{aligned} p_1 &= FALSE \\ p_2 &= TRUE \\ p_3 &= TRUE \\ p_4 &= TRUE \end{aligned}$$

With the above variable assignment all the individual clauses evaluate to TRUE, so by logic their conjunction - the entire Boolean formula - evaluates to TRUE. This simple example of a SAT Problem is solved, and the formula satisfied.

1.2.2 MAX-SAT Problem

Solving a SAT Problem instance is often not possible. In many theoretical as well as real-world instances of the SAT Problem finding the ultimate solution (which results in the problem's Boolean formula evaluating to TRUE), is mathematically impossible. This complication can

be easily illustrated with another very simple SAT Problem example, where the set of variables is

$$X = \{p_1, p_2, p_3\}$$

and the Boolean formula is:

$$(\neg p_1 \vee p_2) \wedge (p_1 \vee p_3) \wedge (p_2 \vee \neg p_3) \wedge (\neg p_1 \vee \neg p_2) \wedge (\neg p_2 \vee \neg p_3)$$

Creating a Boolean table for this formula results in the following:

p_1	p_2	p_3	Clause $\neg p_1 \vee p_2$	Clause $p_1 \vee p_3$	Clause $p_2 \vee \neg p_3$	Clause $\neg p_1 \vee \neg p_2$	Clause $\neg p_2 \vee \neg p_3$	Formula Evaluation (clause conjunction)
0	0	0	1	0	1	1	1	FALSE
0	0	1	1	1	0	1	1	FALSE
0	1	0	1	0	1	1	1	FALSE
0	1	1	1	1	1	1	0	FALSE
1	0	0	0	1	1	1	1	FALSE
1	0	1	0	1	0	1	1	FALSE
1	1	0	1	1	1	0	1	FALSE
1	1	1	1	1	1	0	0	FALSE

As can be seen in the Boolean table, a variable assignment which satisfies the entire formula does not exist, as any configuration of variables results in the formula failing to evaluate to TRUE.

MAX-SAT is an extension of the satisfiability problem which addresses this issue, and provides a mechanic to evaluate variable assignments even if they do not completely satisfy the formula. In a MAX-SAT problem the goal is to determine a variable assignment $v : X \to \{0, 1\}$ which will maximize the amount of satisfied clauses of the formula, even if the formula itself is not completely satisfied. An assignment with the least amount of unsatisfied clauses (known as false clauses) is superior to an assignment with more false clauses. Applying the MAX-SAT mechanic to the previous example lets us differentiate between the variable assignments

p_1	p_2	p_3	Clause $\neg p_1 \vee p_2$	Clause $p_1 \vee p_3$	Clause $p_2 \vee \neg p_3$	Clause $\neg p_1 \vee \neg p_2$	Clause $\neg p_2 \vee \neg p_3$	False Clause Count
0	0	0	1	0	1	1	1	1
0	0	1	1	1	0	1	1	1
0	1	0	1	0	1	1	1	1
0	1	1	1	1	1	1	0	1
1	0	0	0	1	1	1	1	1
1	0	1	0	1	0	1	1	2
1	1	0	1	1	1	0	1	1
1	1	1	1	1	1	0	0	2

Even though no assignment generates a satisfied formula (where the false clause count would be equal to zero) we can determine that some solutions are better than others. For example assignment $\{p1, p2, p3\} = \{0, 1, 0\}$ which results in one false clause is superior to $\{p1, p2, p3\} = \{1, 0, 1\}$ which results in two false clauses. In this paper, any further reference to the SAT Problem actually refers more specifically to the MAX-SAT Problem, as the objective of the algorithms discussed here is to find the lowest possible number of false clauses in a problem, if satisfying it completely is not possible or too difficult.

1.2.3 Motivation

The satisfiability problem is a very popular problem in computer science. By nature, its structure creates many challenges for programmers attempting to find smart algorithms in order to solve the SAT Problems. For years, SAT Competitions take place, where students, researchers and programmers attempt to find the best ways and programs to solve them.

The SAT Problem is much more than just a theoretical problem, however. Its use extends much further than academia, into very applicable fields such as Electronic Design Automation (EDA). Examples of those problems include formal equivalence checking, model checking, formal verification of pipelined multiprocessors or automatic test pattern generation. Modern applications range from termination analysis in term-rewrite systems to circuit-level prediction of crosstalk noise [11].

1.3 Genetic Algorithm

Genetic algorithm (GA) in computer science is a population based stochastic search algorithm, which through intelligent selection and recombination of existing solutions attempts to generate better ones [5]. The concepts of Genetic Algorithms are very strongly based on the principles of biological evolution. In nature, populations of species evolve over time. These species are capable of mutating or crossing over its genes as the evolution progresses. Only the best fit individuals survive, while the weakest ones do not. This process of natural selection can be applied to genetic computation [5].

Specific algorithms applied to the evolutionary learning can greatly vary in how new solutions are introduced into the population (such as in GOMEA-SAT or GASAT approaches), however the main structure of an evolutionary system remains the same.

An initial population is a collection of solutions (known as population members) which is generated before the genetic learning takes place. These population members are (usually) randomly generated, and each represents a possible solution for the given problem. In the case of the SAT Problem, a solution is considered to be a binary sequence, where each bit represents the value for one of the variables of the problem. The fitness measure of how good a given solution is, is also calculated here. In the SAT Problem, fitness refers to the amount of false clauses present in the given solution. The closer the fitness value is to zero, the better the solution.

This is when the evolution process begins. Through selection, recombination and mutation existing solutions are altered in an intelligent way, and new candidate solutions are created. Given a population of solutions selection pressure is applied on this population causing a process of natural selection. Only some subset of solutions is allowed to be picked, forcing the genetic process to work only with better solutions. This has to be done carefully, in order not to loose the diversity in the population. Recombination is a process which happens at every generation step, and results in crossing over two selected solutions (parents), or mutating them, and reintroducing the newly created solution into the population only if it improves it. This process is repeated and over time only the good solutions (ones with the best fitness value) remain.

The details of how new solutions are created, whether it is by recombination with another solution, crossover of multiple solutions, mutation of the current member or a combination of both are algorithm specific and will be explained in later sections of this chapter when some specific algorithms are explained.

At this point in many Genetic Systems the new solution is subject to a local search algorithm, which performs a neighborhood search allowing the new solution to be improved even further by finding its local optimum. This step is not necessary, but often very beneficial.

If the newly created solution satisfies the insertion criteria, meaning it is fit enough to join the population, this new solution replaces one of the previously existing solutions. This process is repeated over time, resulting in the population becoming taken over by better fitted solutions, until a suitable one is discovered.

1.4 GASAT

Genetic Local Search Algorithm for the Satisfiability Problem (GASAT) is one of the hybrid SAT Problem solving algorithms described above. It is based on crossover operators followed by Tabu local search algorithm [7]. The genetic crossover is responsible for finding new, promising starting solutions for local search, while the Tabu search (TS) performs a neighborhood search around this solution in order to optimize the solution [7].

This section describes in more detail the generic concepts and mechanics GASAT uses to accomplish selection, recombination and insertion phases of this evolutionary system.

1.4.1 Selection

In a Genetic Algorithm selection refers to the process of choosing one or more already existing solutions. These candidate solution(s) are subject to genetic alterations which generate a modified candidate solution(s), which can in turn be reintroduced into the population [5]. The specific ways of how GASAT handles this recombination and insertion phases is explained in the next sections.

The selection process itself can become quite complicated, and varies greatly depending on which algorithm is in use. One known technique, which will be used later in this paper, is tournament selection. With tournament selection some number of solutions are first randomly selected from the population. These selected solutions compete against each other and only the best one (one with the best fitness value) becomes selected as a candidate solution during a given tournament.

GASAT does not use tournament selection. Instead it exerts selection pressure directly onto the population. For a given current population P a subpopulation P' is temporarily created. This subpopulation ($P' \subset P$) consists of a fraction p ($0 \leq p \leq 1$) of the best currently existing solutions. With accordance to the GASAT documentation in experiments performed for this paper the value $p = 0.15$, which means that the subpopulation P' consists of the top 15% solutions in the current population [7].

For the recombination stage GASAT needs two candidate solutions. These are randomly chosen from the subpopulation P' [7].

1.4.2 Recombination

The standard evolutionary algorithm approach of modification and recombination of candidate solutions is by performing genetic mutations and crossovers [5]. This can be accomplished in many ways, and many algorithms were created which handle this phase very differently.

GASAT performs a specific crossover operation on two candidate solutions which were chosen during the selection stage. The authors of GASAT put a lot of effort in choosing the most optimal crossover operator for their system. They have determined that using the Collective Clause Crossover (CC) yields best results. This crossover operator takes the two candidate (parent) solutions supplied by the selection process and generates a new, child solution [7].

Corrective Clause Crossover (CC)

This crossover iterates through all clauses looking for ones which are false in both supplied candidate (parent) solutions from the selection process. In order to fix a clause c which is false for parent solutions, flipping one of the variables of the current clause will satisfy it (turn it true).

This very simple solution can be very destructive however, as flipping one variable in order to fix the current clause may affect many other clauses in a negative way. This is why the CC Crossover considers the effect on both parents of flipping each variable for the given clause. For each of the possible flips a collective improvement σ is calculated and compared to other alternatives. In the end, the variable from clause c which causes the greatest collective improvement is flipped [7].

The above process happens only for clauses which are false for both of the original candidate (parent) solutions. For all other variables remaining after this process is completed, the resulting solution takes the variable value of one of the parents with equal probability [7].

Data: two parents X and Y
Result: one child Z
begin
  All the variables of Z are assigned to undefined
  for *each clause* c *such that* $\neg sat(X,c) \wedge \neg sat(Y,c) \wedge \neg sat(Z,c)$ **do**
    for *all positions* i *such that the variable* x_i *appears in* c **do**
      Compute $\sigma = improvement(X,i) + improvement(Y,i)$
    Set $Z|k = flip(X|k)$ where k is the position such that σ is maximum
  All the variables of Z with no value take the value of X or Y with equal probability
end

1.4.3 Local Search

After the Collective Clause Crossover generates a new solution from its parents, this solution is further improved upon. GASAT performs a neighborhood search, in order to explore nearby solutions and find the local optimum for the given solution.

The authors of GASAT decided to use the Tabu Search (TS) as the local search algorithm. The Collective Clause Crossover followed by a SAT Problem optimized Tabu Search results in a powerful search algorithm. The basic ideas behind the TS are explained here.

Tabu Search (TS)

The simple concept behind a Tabu search is to always make the most beneficial move (one that improves the fitness the most, by performing a variable value flip). The best possible move is determined by the same improvement calculation as the one used during the crossover operation [7].

The uniqueness of this search lies in the use of a so called tabu-list. Once the best possible flip is performed the resulting configuration is labeled tabu, and added to the tabu-list. The local search continues searching for another move (variable flip) which causes a maximal improvement, however this flip is only allowed if the resulting binary configuration is not in the tabu-list already. The selection of the best available bit follows this algorithm [7]:

```
Data: an assignment Z, the tabu list tabu, the best assignment found Best
Result: a position
begin
    for all positions i do
        if (Z[i ← flip(Z|i)] ∉ tabu) ∨ (eval(Z[i ← flip(Z|i)]) < eval(Best)) then
            Compute σ = improvement(Z, i)
        else
            σ = −∞
    Return a position that is randomly selected in those which have the maximum σ
end
```

The tabu-list (λ) is always of certain length. At each iteration the newfound configuration is added to the list. Once the tabu-list reaches it capacity, it begins acting as a First In First Out (FIFO) queue, and at every new iteration the oldest solution in the tabu-list is released. The tabu-list (λ) length and the amount of flips (TS iterations) are configurable and provided by an empirical study.

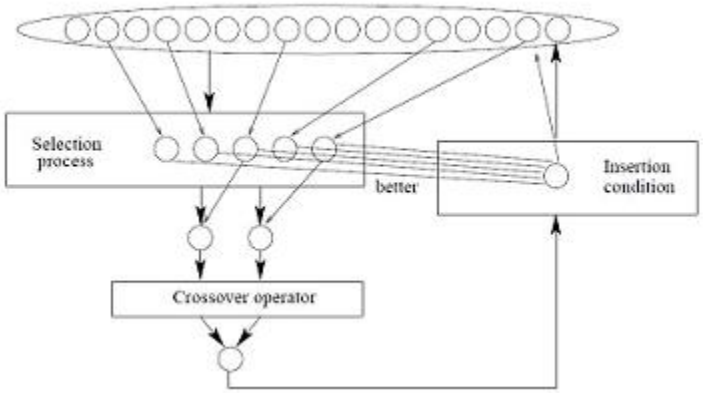
Basically, as the local search continues, best nearby solutions are being found without the risk of falling back into the same local optimum, because the solutions found once are already on the tabu-list, until they are released due to the FIFO mechanism. This allows for a very broad exploration of the neighborhood space.

1.4.4 Insertion

Insertion is the final stage of GASAT's evolutionary step. After two solutions are selected and the crossover operation generates a new solution the Tabu Search attempts to optimize this solution. Now, this solution needs to be inserted back into the population. In GASAT (as well as vast majority of other evolutionary systems) the population size is kept constant. This means that in order for this newly found solution to be inserted back into the population, it needs to replace one of the already existing solutions.

If the new solution does not meet the insertion condition, it is discarded and the population remains unchanged. This insertion condition requires the fitness of this new solution to be better than the worst individual in the current sub-population P' which was created during the selection phase of this generation step. If this insertion condition is met, the new child solution replaces the individual of the population P [7] [8].

The following diagram illustrates the selection, recombination and insertion process [7]:



1.5 Walksat

Walksat is not a Genetic Algorithm. It is a randomized neighborhood search algorithm which is considered to be a very efficient SAT problem solver.

The execution process of Walksat starts with assigning each variable a random binary value. Then the algorithm picks a random clause, out of the ones which are currently not satisfied. Once the clause is selected the algorithm picks a variable which is a part of the clause and flips it [10]. This process is then repeated, and the algorithm keeps track of the improvements caused by the bit flipping. The execution of Walksat is limited only by the amount of bit flips which are allowed during a single execution. It is worth noting that after every bit flip, Walksat needs to recalculate how many false clauses remain in the current solution. This is important because this recalculation can be seen as equivalent to fitness function evaluations for the evolutionary solver approaches. This observation will be useful in creating criteria for comparing the genetic algorithms to Walksat.

A Multi-start Walksat is also used in this paper. Multi-start Walksat simply refers to simultaneous execution of independent Walksat instances on the same SAT Problem. Based on the amount of instances running, the amount of flips allowed for each instance is adjusted so that the total matches a single execution of a one standard Walksat run.

Chapter 2

GOMEA

2.1 LTGA

Linkage Tree Genetic Algorithm (LTGA) is a genetic algorithm which relies on finding variable correlations through a process of building and traversing a linkage tree. Then, the algorithm uses genetic operations in a population based environment in order to optimize existing solutions [13]. Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) is integrated into the Linkage Tree Genetic Algorithm. In this paper the core ideas of GOMEA are extracted from Linkage Tree GA, and applied into the SAT Problem resulting in creation of a new algorithm GOMEA-SAT.

First, the original LTGA will be explained. The next section will show how the core concepts of GOMEA are extracted, applied to the satisfiability problem and how some properties of SAT Problems are used in creating GOMEA-SAT.

The difference between GOMEA and LTGA can be a bit fuzzy. GOMEA can be seen as the genetic algorithm which performs evolutionary operations on population members, based on existence of specific variable subsets. LTGA is an instance of GOMEA which uses Linkage Tree learning in order to generate those clusters. GOMEA-SAT uses a different methodology to generate the clusters. The creation of clusters, and the methods of performing the crossover operations of both LTGA and GOMEA-SAT are explained below.

With LTGA before any genetic crossover operations are performed, the algorithm attempts to learn important information about the problem itself. A Family of Subsets (FOS) is created in this process by learning correlations between different variables present in the problem [13]. Unlike GOMEA-SAT, LTGA accomplishes this by building and traversing a Linkage Tree structure. The aim of using linkage learning is to discover interactions between problem variables. This allows to create subsets (FOS) which are believed to be good building blocks (partial solutions) and should not be disrupted by crossover operations [13]. The official definition of a Linkage Tree used by LTGA is as follows:

Definition: The Linkage Tree of a population of solutions is the hierarchical cluster tree of the problem variables using an agglomerative hierarchical clustering algorithm with a dis-

tance measure D. The distance measure D(X_1,X_2) measures how much dependency there is between two sets of variables X_1 and X_2 [13].

In order to better understand how a Linkage Tree is created, it is important to know how the hierarchical clustering algorithm determines which variables (and sets of variables) to cluster together. This algorithm can be shown in a few steps [12]:

- *Compute the proximity matrix using metric D*
- *Assign each variable to a single cluster*
- *Repeat until one cluster left:*
  - *Join two nearest clusters c_i and c_j into c_{ij}*
  - *Remove c_i and c_j from the proximity matrix*
  - *Compute distance between c_{ij} and all clusters*
  - *Add cluster c_{ij} to the proximity matrix*

After the linkage tree is created, LTGA traverses through the entire population, using each solution along with a randomly selected donor solution (which is also a population member) to create a new solution (child) through a process of copying over parts of the the donor solution. This operation is directly based on the clusters created by the Linkage Tree. These clusters are used as masks in order to create the child solutions. This masking process is identical to the one used in GOMEA-SAT, and will be explained in greater detail in the next section.

Once the child solution is generated, its fitness value is calculated. If the child has a better fitness value then its original, parent solution, the parent is replaced with the child solution, and the traversal of the tree continues. If the child solution in not better than the parent, then the traversal continues without replacing the parent. The LTGA process can be summarized in the following steps [12]:

- *Create initial population of size N*
- *Repeat until stop criteria is met:*
  - *Build Linkage Tree*
  - *For every solution in the population:*
    - *Set crossover mask to current clustering*
    - *Apply the mask from donor to the current solution*
    - *If the offspring solution is better then the parent, replace the parent*
  - *If tree fully traversed: Copy best solution to next population*
  - *Else: continue traversing*

The process of traversing the list of clusters and masking parent solutions is the core of GOMEA and is used in a similar fashion for the GOMEA-SAT. The differences and similarities will be discussed in detail in the next section.

2.2 GOMEA-SAT

Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) has been used as a core part of the Linkage Tree GA. In fact, as mentioned earlier, LTGA can be viewed as an instance of GOMEA with a specific learning mechanism. Evolutionary system created as a part of this thesis, GOMEA-SAT, uses GOMEA as a stand-alone genetic algorithm in order to solve SAT Problems. The FOS structure is created in a very different way than with LTGA, which used Linkage Tree learning [13]. GOMEA-SAT uses the data already present within a SAT Problem to generate the cluster structure (explained in detail in the next section). With this structure GOMEA-SAT performs the evolutionary operations, similar to the ones of LTGA.

Firstly the FOS Structure is determined by a direct variable mapping. Depending on which optimization technique is used, this FOS Structure may be further altered by addition of Singleton subsets. The exact mechanism of creating this structure is explained in the next section. After the FOS Structure is established the initial population of the Evolutionary System is created.

Initial population consists of n randomly generated solutions. Each solution is a binary sequence of length equal to the amount of Boolean variables present in a given SAT Problem, and the location of the bit in the binary sequence represents the current value of the corresponding variable.

For example, a candidate solution for SAT Problem with the Boolean variable set

$$X = \{x_1, x_2, x_3\}$$

could take any of the following bit sequences:

000	001	010	011
100	101	110	111

Once all n population members are randomly initialized, each initial solution is measured with the use of fitness evaluation function. The result of the evaluation function is the fitness of a given solution. This fitness directly corresponds with the amount of false clauses this solution would generate if applied to the given SAT Problem's Boolean formula. Of course the ideal value of the fitness function is zero, which implies satisfying the entire formula and guaranteeing the optimal solution. This is followed by the execution of GOMEA. The goal of GOMEA, and the whole Evolutionary System is to minimize the fitness value as much as possible.

GOMEA examines and attempts to modify each of the n solutions present in the population. In most aspects it very closely follows the GOMEA algorithm used with the Linkage Tree GA [13]. Each of the solutions in the current population is looked at, and considered a candidate solution. One donor solution is picked from the remaining $n - 1$ solutions in the current population. The process of selecting the donor can be random, as in LTGA, however other possibilities such as applying selection pressure with the use of tournament selection were tested.

Once the donor solution is determined, traversing the FOS Structure takes place. Each individual subset in the FOS is considered as a mask, and examined individually. The order of traversing individual subsets has been randomized in order to avoid possible bias towards some subsets. While examining an individual subset, the values of the variables contained within this subset from the donor solution are masked onto the candidate solution, by replacing the original candidate values for those variables. This can be illustrated with a simple example.

Let the set of Boolean variables include five variables,

$$X = \{p_1, p_2, p_3, p_4, p_5\}$$

and the FOS subset currently being examined be

$$s_i = (p_2, p_4, p_5)$$

Lets assume the candidate solution from the population which is currently being examined is 11001 and the Donor solution chosen from the remaining population is 00100. Now a mask needs to be applied from the Donor solution to the Candidate solution for variables p_2, p_4 and p_5, in accordance with the currently examined FOS Subset. This operation will result in:

	Binary Sequence
Candidate	1**1**0**01**
Donor	0**0**1**00**
Result	10000

For each subset, after the donor mask is applied, the new candidate's fitness is calculated. If the new fitness is better than the earlier candidate (before the mask was applied), the new solution becomes the current candidate. After this process is repeated for each FOS subset, the final candidate solution replaces the original solution from the population, unless not a single improvement was possible. In that case, the final candidate solution is discarded and the original solution survives within the population till the next generation step.

This concludes a single evolutionary step of GOMEA-SAT. Depending on the configuration of the Evolutionary System, further evolutionary steps are performed in order to further improve the fitness of existing solutions within the population. The System terminates when either the Boolean formula is completely satisfied (a solution with a fitness of zero is discovered in the population), or when the execution reaches a limit of allowed evolutionary steps. If this limit is reached, the solution with the best fitness present in the population is returned as the result.

2.3 Clause to FOS Mapping

As explained earlier, a Family of Subsets is a structure of clusters of variables. Grouping of those variables in a certain way is very important to the quality of the solution discovered. Before GOMEA is applied in the Linkage Tree GA, this FOS Structure is carefully built. However, building of this linkage tree can be very expensive and the amount of processing

needed in order to create the LTGA's Family of Subsets is a very big drawback.

In the GOMEA-SAT the linkage learning process needed to generate the FOS Structure is replaced with the pre-existing structure which inherently exists in every SAT Problem. This can be accomplished due to the existence of already clustered sets of variables present in a SAT Problem. Recall, the Boolean formula for the SAT Problem is a logical conjunction of a set of clauses:

$$C = \{c_1, c_2, c_3, .., c_m\}$$

resulting in the Boolean formula of form:

$$c_1 \wedge c_2 \wedge c_3 \wedge, .., \wedge c_m$$

where each clause is a disjunction of a subset of variables (or its negations), such as $c_x = (\pm p_a, \pm p_b, \pm p_c, ..., \pm p_k)$, where $a..k$ are some unique variable indexes. With a simple transformation we can use this existing clause structure, and generate a corresponding Family of Subsets:

$$FOS = \{s_1, s_2, ..., s_n\}$$
$$s_x = |c_x|,$$

where s_x is an indexed subset, and c_x is a corresponding clause, so

$$FOS = \{s_1, s_2, ..., s_n\} = \{|c_1|, |c_2|, ..., |c_n|\}$$

It is important to note that in this case the absolute value operator gets rid of the logical negation which could be present within a clause, and the logical disjunction is ignored.

This could be easily illustrated with a simple example. Let:

$$X = \{p_1, p_2, p_3, p_4\}$$

and the Boolean formula be

$$F = (p_1 \vee \neg p_3)^(\neg p_1 \vee p_2 \vee p_3)^(\neg p_1 \vee \neg p_2 \vee p_4)^(p_1 \vee \neg p_3 \vee p_4)$$

This yields four clauses:

$$c_1 = (p_1 \vee \neg p_3)$$
$$c_2 = (\neg p_1 \vee p_2 \vee p_3)$$
$$c_3 = (\neg p_1 \vee \neg p_2 \vee p_4)$$
$$c_4 = (p_1 \vee \neg p_3 \vee p_4)$$

To obtain the FOS Structure, the simple transformation can be applied:

$$FOS = \{s_1, s_2, s_3, s_4\} = \{(|c_1|, |c_2|, |c_3|, |c_4|)\}$$
which yields
$$FOS = \{(p_1, p_3), (p_1, p_2, p_3), (p_1, p_2, p_4), (p_1, p_3, p_4)\}$$

Chapter 3

GOMEA-SAT Experimental Study

3.1 Study Overview

The goal of the first stage of this research is to create an optimal configuration of the evolutionary system for the SAT Problem. Just as the Linkage Tree GA uses GOMEA, so does the GOMEA-SAT. However in order to adapt GOMEA to the Satisfiability problems some alterations need to be made. In order to acquire closer to optimal results from this new algorithm, some changes and additions are implemented and empirically tested on series of benchmarks and configurations. These modifications, which are explained in detail in later sections consisted of testing effects of:

- Population vs. Evolution Length
- Randomizing the FOS Structure
- Singleton additions to the FOS
- Addition of Tournament Selection
- Addition of Forced Improvement

All the additions and configuration modifications were empirically tested on a set of benchmarks acquired from the official SAT Problem on-line library [6]. Specific information about the benchmarks can be found in the Appendix. These benchmarks are divided into three categories:

- Random : where the Boolean Formula and clause variable distribution is randomized
- Handmade : which are specifically designed and implemented manually
- Industrial : which are SAT Problem representations of the real-world problems

By testing benchmarks from each of those categories, the research results are more general and applicable to a much wider range of problems.

3.2 Optimization Techniques and Results

3.2.1 Population vs. Evolution Length

Hypothesis: *With careful adjusting of allowed population size and the amount of generation steps allowed, it will be possible to determine a configuration which will perform statistically better than others with the same amount of processing.*

Intuitively, in an environment without any limitations it would be logical that the more evolutionary steps are allowed during execution of a genetic algorithm, the better the chances of finding a more fit solution. Likewise, the greater the population size of initial solutions, the better the chances of finding more fit solutions over time.

Of course, in reality, GAs are faced with limitations such as time or performance constraints. Given some constrains, a GA needs to be configured correctly. Increasing the population size implies increasing the time and processing needed for every generation step, which in turn results in the need for cutting down on generation steps allowed or the constraint will not be met. Alternatively, increasing the amount of generation steps allowed, forces the decrease of the population size.

A Genetic Algorithm is very susceptible to such adjustments, and trading off too much population space for more generation steps or the other way around can have a very negative effects on the results. Finding a good balance is therefore very important. There are many possible measures which can be used as a constraining factor. Time of execution is a possibility likely used in real-world applications. This is however not a great limiting factor for research purposes, as the same algorithm can have a strongly varying physical runtime based on different hardware/software specifications and conditions of performing the experiments. For this thesis a performance limitation is used as the limiting factor. More specifically - the maximum number of fitness function evaluations is set. With this constraint, it becomes possible to measure and compare how a differently parametrized Evolutionary System behaves purely on algorithmic performance, without the bias of different software or hardware specifications of the machine performing the experiments.

A series of experiments were conducted on three different benchmarks which were parameterized with the following Population / Evolutionary Steps configurations:

- Population: 25 , Evolutionary Steps: 40
- Population: 50 , Evolutionary Steps: 20
- Population: 100, Evolutionary Steps: 10
- Population: 200, Evolutionary Steps: 5
- Population: 500, Evolutionary Steps: 2

All of the above configurations are designed so that each of them performs the same amount of fitness function evaluations, and the differences in results are only based on the

effects of trading off population size for the length of a run and vice-versa. The results of every configuration are gathered and averaged over twenty independent runs.

First benchmark tested was glassyb-v399-s732524269.shuffled-as.sat03-1680.cnf (abbrev. "Glassy"). This benchmark consists of 399 variables used in 1862 clauses.

Population / Evo Steps	False Clause Avr	Std Dev
Glassy 25 / 40	26.0	0.707
Glassy 50 / 20	25.0	1.871
Glassy 100 / 10	25.4	0.548
Glassy 200 / 5	25.8	1.095
Glassy 500 / 2	39.2	2.774

Results of this experiment point to the 50/20 and 100/10 configurations as the most efficient ones. However the significance test was only conclusive in showing that 500/2 configurations is statistically worse. This suggests that too much trade off in the length of the evolution is not desirable, even if it allows for larger populations.

The same experiment was conducted on another benchmark: hidden-k3-s2-r4-n500-01-S1373238829.shuffled-as.sat03-1035.cnf (abbrev. "Hidden"). This benchmark consists of 500 variables and 2000 clauses.

Population / Evo Steps	False Clause Avr	Std Dev
Hidden 25 / 40	17.6	0.894
Hidden 50 / 20	14.4	2.296
Hidden 100 / 10	14.8	2.280
Hidden 200 / 5	19.2	0.837
Hidden 500 / 2	28.4	2.074

Results of this experiment yield more conclusive results. They confirm that the really large populations are bad, if in order to have them evolutionary steps need to be traded off. They also show that having a very large amount of evolution steps, but small population is significantly inferior as well. In this benchmark, the 50/20 and 100/10 configurations were statistically better than any others.

The final benchmark tested is hgen8-n120-03-S1962183220.shuffled-as.sat03-877.cnf (abbrev. "Hgen8"), with 120 variables and 198 clauses.

Population / Evo Steps	False Clause Avr	Std Dev
Hgen8 25 / 40	1.37	0.296
Hgen8 50 / 20	1.27	0.152
Hgen8 100 / 10	1.05	0.229
Hgen8 200 / 5	1.16	0.175
Hgen8 500 / 2	1.84	0.375

Results here have only one statistically superior configuration: 100/10.

The hypothesis is confirmed. While keeping the amount of function evaluations constant, configuring the Population Size versus the length of evolutionary steps is very important. Of course the configurations can be more or less successful based on a given benchmark, however using large populations with little evolutionary steps or vice versa is never desirable, and a good balance needs to be found for a successful EA. For most of further experiments in this paper a population of a hundred solutions and ten generation steps will be used, as it exhibits the most efficient behavior.

3.2.2 Random FOS

Hypothesis: *The usage of Clause to FOS Mapping has a positive effect on the results. Using a randomly generated Family of Subsets will generate significantly worse results.*

Unlike LTGA, in order to greatly speed up the Family of Subsets creation, GOMEA-SAT generates the FOS by some simple transformations to the SAT Problem's clause structure. It is assumed that the very rigid and structured clauses used by the system as building blocks and masks for the genetic operations carry information which are useful for the GA. In order to verify this assumption is correct, and show that the FOS structure created from the Boolean formula truly inherits information useful to the GA process, empirical tests were performed. These experiments tested a FOS structure which is not created from the given SAT Problem, but generated in a random fashion instead. Then, the results of running GOMEA on randomly generated subsets are compared with ones which inherit the structure from the SAT Problem.

The random algorithm used in order to generate random FOS structures was created based on some characteristics of the real SAT benchmarks. For a given problem, the Random FOS algorithm creates the same amount of subsets as the real algorithm would. However, for each individual subset a pseudo-random number of variables is assigned, and each of these variables is randomly picked from X, the set of variables available for the benchmark. This process is then repeated at every step. The term pseudo-random is used here in order to highlight that despite the number of variables within a subset is random (within reasonable range), the average number of variables per subset is forced to become the same as for the real SAT benchmark.

The purpose of enforcing this condition is guaranteeing that the number of fitness evaluations is the same for both, random and correctly generated FOS. This constraint allows for a much better comparison of the two approaches as the computing power is kept the same.

Three benchmarks were used, and the results were verified with the use of the statistical significance test (t-test)

- "Hgen8", with 120 variables and 193 clauses
  (hgen8-n120-03-S1962183220.shuffled-as.sat03-877.cnf)

- "Hidden", with 500 variables and 2000 clauses (hidden-k3-s2-r4-n500-01-S1373238829.shuffled-as.sat03-1035.cnf)
- "Homer19", with 330 variables and 2340 clauses (homer19.shuffled-as.sat03-430.cnf)

Following are results of the experiments and statistical significance tests:

Benchmark	False Cl. Normal	Std Dev Normal	False Cl. Random	Std Dev Random	p-value (T-test)
Hgen8	1.06	0.191	1.60	0.4000	< 0.00001
Hidden	14.25	2.194	27.10	2.171	<0.00001
Homer19	8.00	0.000	8.00	0.000	>0.99999

The results clearly show that if the system is not capable of finding the best solution (it was found in the Homer19 benchmark for both approaches) the Random FOS approach is very inferior. Statistical significance test confirms the hypothesis, and shows that a rigid FOS structure carries advantages over a random and more dynamic one.

3.2.3 FOS Adjustments (Singletons)

Hypothesis: *Adjusting the original FOS structure containing only the existing clauses by adding singleton subsets will significantly improve the results given the same amount of processing allowed.*

So far the FOS structure has been acquired by a direct translation from the SAT Problem's clause structure. Recall, the clause masks used to create candidate solutions during the genetic steps are directly related to each existing subset. This results in no mask being of lesser length than the shortest clause in the SAT problem. This could potentially be a problem.

If, on average, the clause length of a given SAT Problem is relatively large, the building blocks used to mask the donor solution onto the candidate are also large. This could be problematic as consistently masking large subsets could be failing to generate potentially good solutions which could only be found using smaller modifications.

In order to counteract this unwanted potential effect, a modification to the FOS structure is proposed in this section. Normal FOS Structure generated from SAT Problem's Boolean formula remains intact, however it is now appended with a new set of subsets - Singletons. The singleton subsets are generated directly from X, the variable set of the SAT Problem. Recall each SAT problem consists of such set:

$$X = \{x_1, x_2, x_3, .., x_n\}$$

the singleton subsets are a direct one-to-one translation of X, which generates a set of singleton subsets of length one:

$$FOS_{singleton} = \{p_1, p_2, p_3, .., p_n\} = \{x_1, x_2, x_3, .., x_n\}$$

The new extended subset structure (FOS_{ext}) is simply the union of the originally created FOS and the singleton addition (FOS_{sin})

$$FOS_{ext} = FOS_{sin} \cup FOS$$

With this extended FOS structure the genetic algorithm is now capable of exploring what happens when only very small changes are done to the candidate solution, by not only masking clauses with many variables but also by applying single bit masks.

In order to verify if this approach has a positive effect on the GA results, more modifications to the GA process are necessary. Because of extending the FOS structure with the new subsets more function evaluations will be needed for every examined solution. The amount of additional evaluations needed is unique to a individual SAT Problem as it depends on how many variables a given problem has, with respect to how many clauses are present. In order to account for this, for each problem the fraction representing the increase of evaluations needed is calculated. This fraction is then taken away from the original amount of evolutionary steps allowed for the given experiment. This way, an extended FOS performs more evaluations within a single evolutionary step, however it is allowed proportionally less steps. Thanks to this modification, the number of fitness function evaluation remains the same, and allows for non bias comparison of the GA using either the standard or the singleton extended FOS Structures.

Three SAT Problem benchmarks are used here to evaluate the effects of replacing the FOS Structure, with the FOS_{ext} Structure defined earlier. These benchmarks are:

- "Hgen8", with 120 variables and 193 clauses (hgen8-n120-03-S1962183220.shuffled-as.sat03-877.cnf)
- "Hidden", with 500 variables and 2000 clauses (hidden-k3-s2-r4-n500-01-S1373238829.shuffled-as.sat03-1035.cnf)
- "Homer19", with 330 variables and 2340 clauses (homer19.shuffled-as.sat03-430.cnf)

Following are results of the experiments:

Benchmark	False Cl. (FOS)	Std Dev (FOS)	False Cl. (FOS_{ext})	Std Dev (FOS_{ext})	p-value (T-test)
Hgen8	1.06	0.191	1.09	0.302	>.9999
Hidden	14.25	2.194	15.25	1.861	0.2354
Homer19	8.00	0.000	8.00	0.000	>.9999

After examining the results it is clear that the hypothesis of this section is not correct. Performing the significance test (t-test) does not show any statistically significant improvement of using FOS_{ext} over regular FOS. It is however important to note that this result does not render using the extended FOS useless. The results of both methods are equivalent, as the same amount of function evaluations is enforced, and neither result is significantly worse than the other. In this paper I will continue using the standard FOS method, as the FOS_{ext} creates a small additional overhead, without improving the results.

3.2.4 Tournament Selection

Hypothesis: *Using a size s=2 or s=3 Tournament Selection in the process of determining the donor solution will produce significantly better results than choosing the donor randomly.*

Selection Pressure is a characteristic of many genetic algorithms. This characteristic is used to describe how a parent solution is determined during the selection stage. For GOMEA such selection pressure could be applied to the process of choosing the donor solution, which is then used for masking the candidate solutions. Original GOMEA does not apply any pressure to the donor selection process and simply randomly picks one of the existing solutions within the current population [13]. In this section selection pressure will be applied to the donor solution, specifically Tournament Selection.

Tournament Selection is a known evolutionary algorithm technique used for applying selection pressure to the population. Instead of simply choosing one solution to use as a parent (or a donor in GOMEA case), a pool of solutions of size n (s = n) is picked. Only the best solution from this pool will be allowed to become selected. The goal of this technique is to slightly increase the probability of choosing a better solution as a parent/donor, instead of giving equal chances to all solutions, despite how fit they are [9].

For GOMEA-SAT two variants of the Tournament Selection were implemented and tested. The first one, s=2, would pick two solutions from the populations, and only allow the better one to become the donor. The second one, s=3, works analogically and chooses the best out of three randomly picked solutions as a donor.

Two sets of series of experiments (for s=2 and s=3) are performed of the following SAT Problem benchmarks:

- "Hgen8", with 120 variables and 193 clauses
  (hgen8-n120-03-S1962183220.shuffled-as.sat03-877.cnf)
- "Hidden", with 500 variables and 2000 clauses
  (hidden-k3-s2-r4-n500-01-S1373238829.shuffled-as.sat03-1035.cnf)
- "Homer19", with 330 variables and 2340 clauses
  (homer19.shuffled-as.sat03-430.cnf)
- "Genurq4", with 64 variables and 298 clauses
  (genurq4Sat.shuffled-as.sat03-1510.cnf)

At first the s=2 case is examined. The values in the columns represent the average amount of false clauses and their standard deviations for the no tournament case and the t=2 case. Last column, p-value, is a statistical significance test indicator. It is a standard assumption that the p-value needs to be smaller than 0.05 on order for there to be a statistically significant difference.

Benchmark	Normal (No Tournament)	Std Dev (No Tournament)	Tournament s=2	Std Dev s=2	p-value (T-test)
Hgen8	1.06	0.191	1.07	0.257	0.9294
Hidden	14.25	2.194	15.55	1.238	0.1250
Homer19	8.00	0.000	8.00	0.000	>.9999
Genurq4	0.35	0.489	0.30	0.470	0.7436

Tournament with three possible donors is examined next (s=3). Format of the data is identical to the size = 2 tournament explained above.

Benchmark	Normal (No Tournament)	Std Dev (No Tournament)	Tournament s=3	Std Dev s=3	p-value (T-test)
Hgen8	1.06	0.191	1.04	0.190	0.8158
Hidden	14.25	2.194	14.15	1.424	0.9054
Homer19	8.00	0.000	8.00	0.000	>.9999
Genurq4	0.35	0.489	0.35	0.489	>.9999

Results of the experiments on both Tournament Selection sizes are very conclusive. Neither attempt of improving the results by forcing the possible donor solutions to compete against each other has any statistically significant effect on the results. This can be observed by examining the T-test's p-values, which are very high. While adding the Tournament Selection can be extremely useful in many Genetic Algorithms, it does not cause any improvement for the GOMEA. The hypothesis of this section does not hold and the addition of Tournament Selection to the GOMEA-SAT would only create an unnecessary overhead.

3.2.5 Forced Improvement

Hypothesis: *Addition of the Forced Improvement mechanism to the GOMEA Algorithm will generate statistically significant improvements with the same amount of processing allowed.*

Forced Improvement (FI) is a mechanism which changes the behavior of GOMEA in certain cases in order to attempt to force-find a better solution when the unmodified GOMEA cannot. The FI mechanism is triggered only when GOMEA traversed the whole FOS structure of a given candidate population member, but none of the masks created from the donor solution succeed in improving this particular solution. If the standard algorithm finds even the slightest improvement by itself, FI is not triggered [1].

Recall that in the standard GOMEA the solution being examined remains unchanged in the current population if the building block masks are not capable of improving it. The same solution will be again present in the next step's population, and a different donor solution will attempt to improve it.

Forced Improvement manages the scenario described above differently. Instead of leaving the solution unchanged in the population and allowing the next generation to handle it, FI

attempts to fix it. The best currently available solution is located in the population. This solution becomes the new donor, and the masking process is repeated until a single improvement is found or all the masks from the new donor are tried and still no improvement is possible [1].

A very important aspect of Forced Improvement is that traversing and applying the masks of the best fit donor only happens until any single mask improves the candidate solution. This is very different than the standard GOMEA process, where even after a mask improves the candidate solution, the change is kept with the candidate and the process continues until all of the subsets have been masked [12]. The reason for this difference is very important. Allowing a solution to be fully masked with the current best solution in the population every time the original donor was not able to find an improvement affects the diversity of the population in a strong and negative way. Applying all the masks of the same same solution, even a very good one, to many solutions in the population is likely to force the population to converge towards a false optimum.

The goal of the Forced Improvement is not to force the best solution onto the candidate solution, but to use this good solution in order to find a very small improvement on a candidate solution which a random donor could not improve. With a very small improvement to the candidate solution the population will not start losing diversity, but at the next generation step the regular GOMEA process will have an easier job improving the new candidate solution.

A downside to the Forced Improvement mechanism is that it requires more processing and more function evaluations within a single generation step. This in turn results in shortening the evolutionary steps proportionally.

This mechanism was tested with the following SAT Problem benchmarks:

- "Hgen8", with 120 variables and 193 clauses
  (hgen8-n120-03-S1962183220.shuffled-as.sat03-877.cnf)
- "Hidden", with 500 variables and 2000 clauses
  (hidden-k3-s2-r4-n500-01-S1373238829.shuffled-as.sat03-1035.cnf)
- "Homer19", with 330 variables and 2340 clauses
  (homer19.shuffled-as.sat03-430.cnf)
- "Homer17", with 286 variables and 1742 clauses
  (homer17.shuffled-as.sat03-430.cnf)
- "Glassy", with 399 variables and 1862 clauses
  (glassyb-v399-s732524269.shuffled-as.sat03-1680.cnf)

The following table presents the results of measuring the average false clauses (f.c.) on the SAT Problem benchmarks for the regular GOMEA approach as well as the GOMEA with the built-in Forced Improvement mechanism. For the approach using FI, maximum evolutionary steps allowed has been shortened proportionally to the average of additional evaluations

needed in a solution. P-value is again the indicator of statistical significance.

Benchmark	False Clauses (No FI)	Std Dev (No FI)	False Clauses (with FI)	Std Dev (with FI)	p-value (T-test)
Hgen8	1.06	0.191	1.11	0.310	0.6694
Hidden	14.25	2.194	15.30	2.145	0.2344
Homer19	8.00	0.000	8.00	0.000	>.9999
Homer17	4.00	0.000	4.00	0.000	>.9999
Glassy	28.2	2.049	25.2	1.549	0.002

The hypothesis is confirmed. Adding Forced Improvement to the GOMEA process doesn't always significantly improve results, however it does accomplish that in some cases. At the same time, the FI addition does not create significantly worse results in any of the tested cases, which implies that Forced Improvement does in fact improve GOMEA-SAT.

3.2.6 Conclusions

The Gene-pool Optimal Mixing Evolutionary Algorithm adapted for the Satisfiability Problem was extensively tested in this part of the research. The goal of this testing was to determine and examine configurations, modifications and extensions to GOMEA which would result in optimizing the results of this algorithm. Some of this research resulted in not finding any improvements and sometimes even worsening the outcome, however a few approaches generated very promising results.

GOMEA-SAT benefits from the fact that its Family of Subsets structure is easily derived from the existing clause structure of the Satisfiability problem. One premise of this thesis relied on the assumption that even if not ideal, the use of SAT clauses as building blocks for the genetic processes will carry enough information and stability to be a very good alternative for generating and using the FOS structure. This hypothesis was confirmed. Testing the same SAT Problem benchmarks on a more dynamic and randomly generated subsets turned out to be very statistically inferior.

Genetic Algorithms are very sensitive to how some of their parameters are configured. Many configurations were tested in order to determine a good combination of population size and the amount of evolutionary steps a GA should be able to take. Of course, increasing population size along with increasing the number of steps is likely to generate better results. This isn't always possible as there are constraints on the execution of a genetic algorithm. With some constraints in place, increasing population size forces a trade off in evolution length, and vice versa. Some configurations tested a lot better than others, allowing to find the parameters generating much better results.

There was some concern that the building blocks generated from the SAT Problem's clause structure might affect the creation of new solutions too strongly. Extending the FOS structure with a set of singletons (FOS_{sin}), which consisted of subsets of length one was theorized to improve the results. While the use of extended FOS structure did not have a negative effect

on the results, it also did not improve them.

Another failed approach of improving the results was the application of selection pressure. A Tournament Selection mechanism was introduced to the genetic algorithm. The goal was to force the donor solution to be more fit than one chosen randomly, with the hope of generating better results. This hypothesis was also proven wrong, as both of the tournament sizes tested (s=2 and s=3) had virtually no significant effects on the outcome.

The Forced Improvement mechanism turned out a lot more useful. By giving an unimproved candidate solution another chance at improvement within the same generation step, some significant improvements were recorded, making the integration of FI into GOMEA-SAT a welcome addition.

The goal of this section was to determine the correct configurations and useful additions and modifications to the pure genetic algorithm. But even an optimized GOMEA algorithm is not yielding results competitive with some other SAT Solving methods. Next section will address this by introducing a neighborhood search into the GOMEA-SAT in hopes of making it much more powerful.

Chapter 4

Local Search

Hypothesis:*Addition of the short local search (LS) with every new solution will greatly improve the previous experiments, and will result in creating a system significantly competitive with, or better than GASAT, single long-run Walksat and multi-start Walksat Algorithms.*

As introduced earlier, Walksat is a neighborhood search algorithm designed for solving satisfiability problems. As any local search Walksat performs a series of small changes to the solution exploring the nearby solutions in order to determine the optima [10]. It is designed to perform a big scale bit modifications in order to find optimal SAT Problem solutions. It is a very efficient and effective algorithm.

Walksat performs a great amount bit flips as it has no other mechanism to generate solutions. GOMEA-SAT on the other hand can easily move around the solution space, however it often takes big, and not optimized leaps. This section will focus on incorporating the power of a local search into the GOMEA-SAT created in the previous sections.

Every new candidate solution inserted into the population is better than its predecessor, however it is incapable of knowing if an even better solution exists somewhere very near in the solution space. With an addition of a small local search to any newly found solution, the population will be updated not with just a better solution, but instead with an even better local optimum result surrounding the solution.

In order to verify that using a GA in combination with the Walksat local search is desirable, the final results will be compared with a Multi-start Walksat Algorithm as well as the pure Walksat. All three methods will be explained in detail in the following section.

4.1 Local Search Integration

Part of incorporating a local search into the GA requires determining how long for every new solution is the local search allowed to run while searching for a local optimum. Three cases are considered:

- Small LS

- Medium LS
- Large LS

Each case has a different cutoff value for flips allowed. Small local search only allows 100 bit flips for every solution in the population per generation step. Medium local search can perform 20,000 flips, while the Large search is allowed to flip up to 40,000 bits. Each of these LS configurations are tested over a range of benchmarks.

As GOMEA-SAT performs an evolutionary step every solution in the population is looked at. With the use of the FOS structure every solution is attempted to be improved upon. If this process succeeds a new solution replaces the original one. If it does not succeed, the original solution remains in the population. With the local search integration into the GA process, weather a new solution is introduced or the original remains, a Walksat local search will be performed before advancing to the next solution.

4.2 Walksat, Multistart Walksat and GASAT

A measure of GOMEA-SAT is needed to determine if the use of this genetic algorithm in combination with the local search creates a new, viable alternative to solving SAT Problems.

Walksat is already a stand-alone powerful SAT Problem solver. It can be used as a great comparison tool in order to see how much impact the evolutionary learning aspect of GOMEA-SAT really has. The amount of total local search flips performed by the GOMEA-SAT's bursts of Walksat LS can be easily calculated:

$$Flips_{total} = Cutoff * PopulationSize * NumberofEvolutionarySteps$$

By allowing the stand-alone Walksat to perform $Flips_{total}$ bit flips and comparing the results to GOMEA-SAT it will be possible to determine how much impact the genetic algorithm really has.

Another measure of determining the usefulness of GOMEA-SAT is to compare it against a Multi-start Walksat. In a sense, GOMEA-SAT keeps restarting a local search on each of its solutions. However in between of these searches, the evolutionary learning takes place which improves the solutions before the local search is triggered again. If this intelligent learning part is removed, all that remains is a Multi-start Walkat algorithm. Total bit flips performed by the local search if GOMEA-SAT can be split into independent Multi-start runs. The comparison between GOMEA-SAT and this Multi-start Walksat should also highlight how much does the evolutionary system affect the results.

The same set of benchmarks is used for the experimental results sections below, and consists of:

- "Hgen8", with 120 variables and 193 clauses
  (hgen8-n120-03-S1962183220.shuffled-as.sat03-877.cnf)

- "Hidden", with 500 variables and 2000 clauses (hidden-k3-s2-r4-n500-01-S1373238829.shuffled-as.sat03-1035.cnf)
- "Genurq4", with 64 variables and 298 clauses (genurq4Sat.shuffled-as.sat03-1510.cnf)
- "2nc", with 2756 variables and 10886 clauses (2000009987nc.shuffled-as.sat03-1665.cnf)
- "Glassy", with 399 variables and 1862 clauses (glassyb-v399-s732524269.shuffled-as.sat03-1680.cnf)

4.3 Small LS Results

This section looks into the smallest cutoff value for the local search. Every solution after being examined by GOMEA will perform a short Walksat with a cutoff of 100 flips. All the benchmarks are parametrized with a population of one hundred solutions and five evolutionary steps are allowed.

Using the $Flips_{total}$ formula this implies that the stand-alone Walksat will be permitted to run for $5 * 10^4$ flips, while the Multi-start Walksat will have ten start points each allowed $5 * 10^3$ flips.

Benchmark	GOMEA-SAT (with LS) false cl. avr	Stand alone Walksat false cl. avr	Multi Start Walksat false cl. avr
Hgen8	1.00	1.00	1.00
Hidden	4.40	0.00	0.00
Genurq4	0.00	0.00	0.00
2nc	128	94.8	248
Glassy	17.9	7.80	16.5

The significance test results of GOMEA-SAT versus the two Walksat configurations are summarized in the following table:

Benchmark	GOMEA-SAT (with LS) vs. Stand alone Walksat p-value (T-test)	GOMEA-SAT (with LS) vs. Multi Start Walksat p-value (T-test)
Hgen8	>.9999	>.9999
Hidden	<.0001	<.0001
Genurq4	>.9999	>.9999
2nc	<.0001	<.0001
Glassy	<.0001	0.1422

The results do not look promising for the evolutionary system. GOMEA-SAT never gains any statistical advantage over the other two algorithms, moreover it is statistically worse on a couple occasions (Glassy and Hidden benchmarks). This implies that using a very small local search within the GOMEA-SAT is in no way beneficial, and a pure local search is likely to outperform it.

4.4 Medium LS Results

The cutoff is greatly increased here. Every solution within the population is now allowed $2 * 10^4$ local search flips at every generation step. The other parameters remain the same with population of one hundred and five evolutionary steps.

In the same fashion as in the previous section the $Flips_{total}$ for stand alone Walksat is calculated to be 10^7 flips, and the Multi-start Walksat is allowed 10^6, while having 10 different parallel starting points.

After conducting the experiments, the results are as follows:

Benchmark	GOMEA-SAT (with LS) false cl. avr	Stand alone Walksat false cl. avr	Multi Start Walksat false cl. avr
Hgen8	1.00	1.00	1.00
Hidden	0.00	0.00	0.00
Genurq4	0.00	0.00	0.00
2nc	44.7	50.0	60.1
Glassy	5.80	5.50	6.00

The statistical significance results are as follows:

Benchmark	GOMEA-SAT (with LS) vs. Stand alone Walksat p-value (T-test)	GOMEA-SAT (with LS) vs. Multi Start Walksat p-value (T-test)
Hgen8	>.9999	>.9999
Hidden	>.9999	>.9999
Genurq4	>.9999	>.9999
2nc	0.3092	<.0001
Glassy	0.1974	0.2140

Examining these results yields very different conclusions than the previous section. GOMEA-SET is no longer statistically worse than any benchmark with respect to both Stand-alone and Multi-start Walksat. Moreover, in one instance it significantly outperforms the Multi-Start local search. Keeping the previous section's result in mind, it is becoming more clear that a longer local search is beginning to work in favor of the evolutionary system.

4.5 Large LS Results

In this section a very large local search is allowed within the evolutionary system. The cutoff becomes $4 * 10^4$ which analogically to the previous sections allows for $2 * 10^7$ flips for the Stand-alone Walksat, and $2 * 10^6$ flips for the ten restarts of the Multi-start Walksat each.

This computationally heavy configurations generate the following results:

Benchmark	GOMEA-SAT (with LS) false cl. avr	Stand alone Walksat false cl. avr	Multi Start Walksat false cl. avr
Hgen8	1.00	1.00	1.00
Hidden	0.00	0.00	0.00
Genurq4	0.00	0.00	0.00
2nc	38.4	47.7	58.0
Glassy	5.50	5.6	6.00

Statistical analysis on these experiments results in the following:

Benchmark	GOMEA-SAT (with LS) vs. Stand alone Walksat p-value (T-test)	GOMEA-SAT (with LS) vs. Multi Start Walksat p-value (T-test)
Hgen8	$>.9999$	$>.9999$
Hidden	$>.9999$	$>.9999$
Genurq4	$>.9999$	$>.9999$
2nc	$<.0001$	$<.0001$
Glassy	$>.9999$	$>.9999$

These results are even more in favor of the GOMEA-SAT. With respect to the Multi-start system, the evolutionary algorithm either significantly outperformed it, or both managed to find the globally optimal solution. Stand-alone Walksat was also either statistically inferior or the same as GOMEA-SAT, but never managed to be better anymore.

4.6 GASAT Comparison

Previous results show that GOMEA-SAT is most efficient when a larger local search is allowed. GOMEA-SAT can be considered a hybrid algorithm, as it combines genetic manipulation with a neighborhood search. Another hybrid algorithm which has been described earlier in this paper, GASAT, also attempts to solve the same problem. GASAT is a known and successful genetic algorithm specifically designed to solve the satisfiability problem [7] [8].

GASAT uses a very different method of genetic alteration, and a very different neighborhood search method. While GOMEA-SAT uses Walksat LS, GASAT uses Tabu search, as explained in detail in the introductory section. They are, however, both genetic algorithms designed to solve the same problem.

In order to make a fair comparison of the performance of the two algorithms some limitations are put in place, and the two algorithms are tested against each other on the same benchmarks. The amount of function evaluations is the same, as is the number of flips allowed during the local search phase. The population size n is also kept the same at $n = 100$.

In order to compare the two algorithms the following benchmarks are used:

- "Hgen8", with 120 variables and 193 clauses (hgen8-n120-03-S1962183220.shuffled-as.sat03-877.cnf)

- "Hidden", with 500 variables and 2000 clauses (hidden-k3-s2-r4-n500-01-S1373238829.shuffled-as.sat03-1035.cnf)
- "Genurq4", with 64 variables and 298 clauses (genurq4Sat.shuffled-as.sat03-1510.cnf)
- "2nc", with 2756 variables and 10886 clauses (2000009987nc.shuffled-as.sat03-1665.cnf)
- "Glassy", with 399 variables and 1862 clauses (glassyb-v399-s732524269.shuffled-as.sat03-1680.cnf)

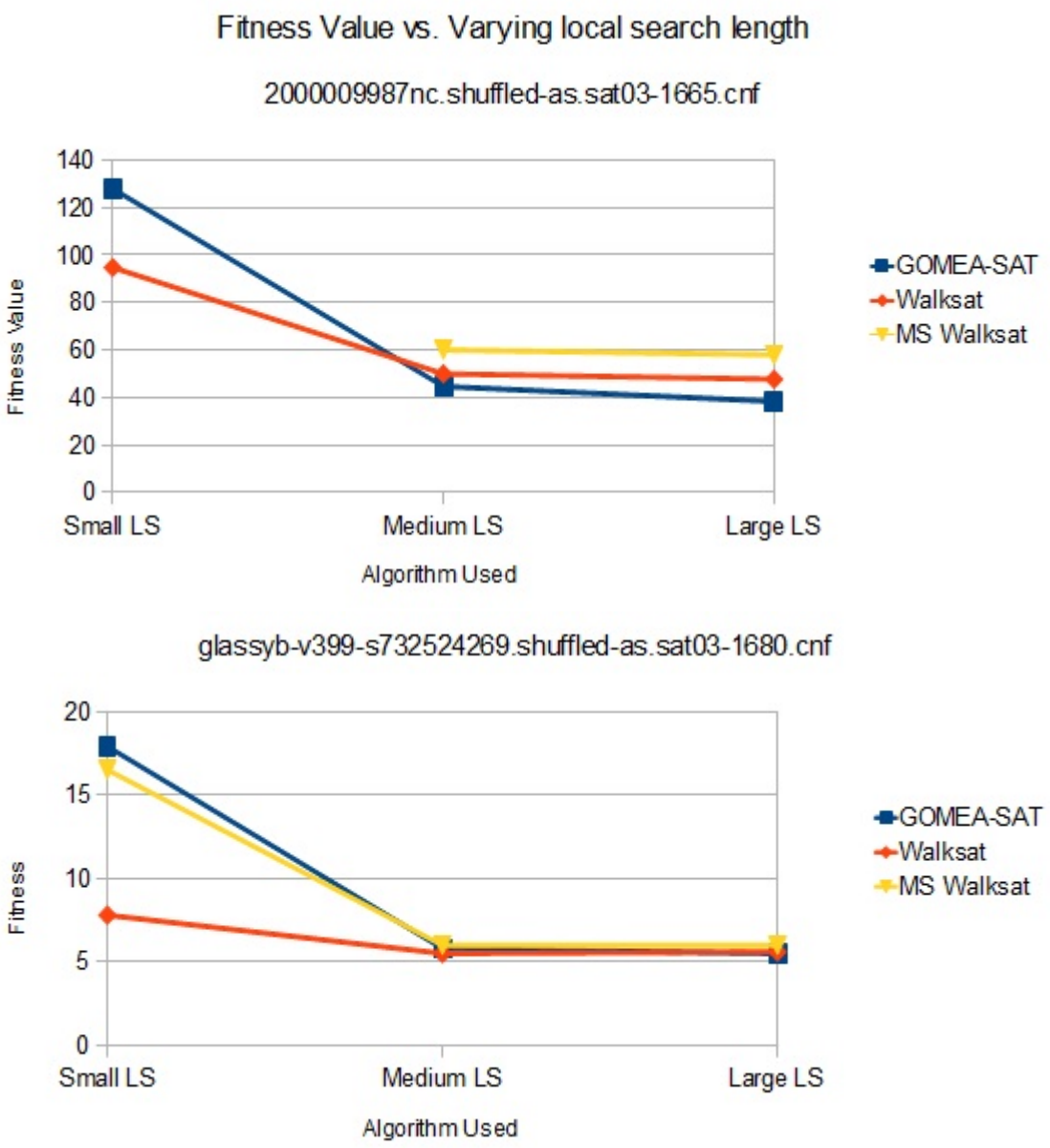
The empirical results are as follows:

Benchmark	GOMEA-SAT (with LS) false cl. avr	GASAT false cl. avr	p-value T-Test
Hgen8	1.00	1.02	>.9999
Hidden	0.00	0.00	>.9999
Genurq4	0.00	0.00	>.9999
2nc	38.4	51.3	<.0001
Glassy	5.50	5.25	0.5990

These results also confirm the hypothesis. GOMEA-SAT performance is not significantly inferior to the one of GASAT. In most of the tested benchmarks GOMEA-SAT performs just as well as GASAT, and even significantly outperforms it on one occasion.

4.7 Results Conclusion

Adding a local search into the evolutionary system created for this thesis very significantly improved the system. However, the GOMEA-SAT with the local search doesn't always look good when contrasted against pure local searches. It is very clear that the overhead carried by the evolutionary operations is actually disruptive when the local search bursts are very small.



The two graphs presented here show how much effect the amount of local search has on the different algorithms tested. These two benchmarks were chosen as they presented the most statistically significant results.

In most cases Stand-alone and Multi-start Walksat both easily outperformed GOMEA-SAT when the duration of the local search was short, despite those two algorithms being much simpler and faster. However when the duration of the local searches are made larger, the advantages of genetic computing emerge. Pure Walksat starts to lose its advantage, and the intelligent learning of the genetic system starts to pay off. This becomes most clear on the longest tested run, when the genetic properties of GOMEA-SAT are clearly the reason why pure Walksat is statistically equal and even significantly outperformed at times by the

GA, while allowed the same amount of local search flips. It can be seen from the graphs that the GOMEA-SAT gains advantage as the local search becomes long.

In comparison with another hybrid local search genetic algorithm - GASAT, GOMEA-SAT also performs very well. Most of the time it is not able to outperform GASAT, however it is never statistically significantly worse, which shows a lot of promise in further development of this approach.

Chapter 5

Learning

5.1 Overview

One of the main features of GOMEA-SAT is a unique way of generating the Family of Subsets. Recall that this operation is done by a simple mapping from the inherent structure of a SAT Problem's Boolean formula onto the FOS clause structure. This transformation is very efficient as far as time complexity ($O(n)$) and has proven successful in creating a competitive SAT Problem solver. It might however not be the optimal FOS creation method.

The goal of this chapter is to focus on the FOS structure creation and examine if applying dynamic learning to this structure can benefit GOMEA-SAT and generate statistically better results. In order to test this hypothesis, the mapping of Boolean formula clauses onto the FOS will be completely replaced by the Linkage Tree learning algorithm, which was discussed in earlier chapters, and will be adapted for the Satisfiability problem.

This means that a Linkage Tree will be created for the SAT Problem instance, and the correlations of variables (and sets of variables) will be kept and recalculated at every generation step. As mentioned earlier the Linkage Tree creation is expensive and is most likely too computationally heavy for many large SAT benchmarks. The goal of this chapter is not to create a much better solver than the GOMEA-SAT discussed so far, but to verify if addition of Linkage Tree learning to that system can have a positive effect.

The experiments are ran with the same conditions and parameters, despite the fact that the use of Linkage Trees is much heavier computationally than the original SAT Problem to FOS mapping. The purpose of that is to see how much the learning of structure can influence the final results.

In order to accomplish this, the LTGA (Section 2.1) is modified in order to be compatible with satisfiability problem benchmarks. The results are then compared to the GOMEA-SAT algorithm running under the same conditions, but without the Linkage Tree structure learning mechanism.

5.2 Configuration

In order to make GOMEA-SAT comparable to LTGA the amount of function evaluations for each algorithm needs to be the same for a given benchmark. This constraint is not completely straightforward to implement as the size of the Family of Subsets differs based on the benchmark and algorithm used.

The amount of function evaluations performed by GOMEA-SAT is

$$Evals_{GS} = c * n_{GS} * s_{GS}$$

where c is the number of clauses of the Boolean formula, n_{GS} is the population size and s is the amount of generation steps allowed.

For LTGA, the number of evaluations performed is

$$Evals_{LT} = (2l - 2) * n_{LT} * s_{LT}$$

where l is the length of the solution bit string (the amount of variables present in a given SAT Problem), n_{LT} is the size of the population and s_{LT} is the amount of generation steps.

In order to keep the allowed amount of function evaluations equal between the two algorithms, the length of evolution for LTGA (amount of evolutionary steps s_{LT}) will be adjusted:

$$Evals_{LT} = Evals_{GS}$$
$$(2l - 2) * n_{LT} * s_{LT} = c * n_{GS} * s_{GS}$$

Evaluating for s_{LT} leads to:

$$s_{LT} = \frac{c*n_{GS}*s_{GS}}{(2l-2)*n_{LT}}$$

Some of these variables are already known, as the GOMEA-SAT experiments were performed when $n_{GS} = 100$ and $s_{GS} = 10$. This simplifies the formula:

$$s_{LT} = \frac{500c}{(l-1)*n_{LT}}$$

For the experiments performed in the next section, three different values of n_{LT} are used in order to allow a balance of population size and evolution length while keeping the amount of function evaluations consistent. The n_{LT} values used are 100, 250 and 500, which result in three different evolution lengths $s_{LT_{100}}, s_{LT_{250}}, s_{LT_{500}}$:

$$s_{LT_{100}} = \frac{5c}{l-1}$$
$$s_{LT_{250}} = \frac{2c}{l-1}$$
$$s_{LT_{500}} = \frac{c}{l-1}$$

As explained earlier the values c (amount of clauses of a SAT Problems Boolean formula) and l (the amount of variables of a SAT Problem) vary depending on the benchmark tested. In the experiments, the above formulas were applied in order to modify the amount of evolutionary steps of LTGA. This way, the amount of function evaluations for a given benchmark is kept equal between GOMEA-SAT and LTGA, while different population sizes are tested.

5.3 Results

Performance of LTGA against GOMEA-SAT is tested on the following six benchmarks:

- "Hgen8", with 120 variables and 193 clauses
  (hgen8-n120-03-S1962183220.shuffled-as.sat03-877.cnf)
- "Hidden", with 500 variables and 2000 clauses
  (hidden-k3-s2-r4-n500-01-S1373238829.shuffled-as.sat03-1035.cnf)
- "Homer19", with 330 variables and 2340 clauses
  (homer19.shuffled-as.sat03-430.cnf)
- "Homer17", with 286 variables and 1742 clauses
  (homer17.shuffled-as.sat03-430.cnf)
- "Genurq4", with 64 variables and 298 clauses
  (genurq4Sat.shuffled-as.sat03-1510.cnf)
- "Glassy", with 399 variables and 1862 clauses
  (glassyb-v399-s732524269.shuffled-as.sat03-1680.cnf)

The results are as follows for GOMEA-SAT versus LTGA ($s_{LT_{100}}$), LTGA ($s_{LT_{250}}$) and LTGA ($s_{LT_{500}}$) respectfully :

Benchmark	GOMEA-SAT false cl. avr.	GOMEA-SAT st. dev.	LTGA ($s_{LT_{100}}$) false cl. avr.	LTGA ($s_{LT_{100}}$) st. dev.	p-value (T-test)
Hgen8	**1.06**	0.191	1.50	0.527	0.0008
Hidden	**14.25**	2.194	21.00	3.391	<.0001
Homer19	8.00	0.000	8.00	0.000	>.9999
Homer17	4.00	0.000	4.00	0.000	>.9999
Genurg4	**0.35**	0.489	1.00	0.000	<.0001
Glassy	**28.2**	2.049	30.90	2.508	0.0002

Benchmark	GOMEA-SAT false cl. avr.	GOMEA-SAT st. dev.	LTGA ($s_{LT_{250}}$) false cl. avr.	LTGA ($s_{LT_{250}}$) st. dev.	p-value (T-test)
Hgen8	1.06	0.191	1.00	0.000	>.9999
Hidden	**14.25**	2.194	18.80	1.483	<.0001
Homer19	8.00	0.000	8.00	0.000	>.9999
Homer17	4.00	0.000	4.00	0.000	>.9999
Genurg4	0.35	0.489	0.579	0.507	0.1676
Glassy	28.2	2.049	**26.20**	1.304	0.0008

Benchmark	GOMEA-SAT false cl. avr.	GOMEA-SAT st. dev.	LTGA ($s_{LT_{500}}$) false cl. avr.	LTGA ($s_{LT_{500}}$) st. dev.	p-value (T-test)
Hgen8	1.06	0.191	1.10	0.316	>.9999
Hidden	**14.25**	2.194	20.40	2.701	<.0001
Homer19	8.00	0.000	8.00	0.000	>.9999
Homer17	4.00	0.000	4.00	0.000	>.9999
Genurg4	0.35	0.489	0.36	0.495	0.9466
Glassy	28.2	2.049	27.20	0.837	0.0538

The results show that with equal amount of function evaluations the Linkage Tree learning fails to significantly improve the GOMEA-SAT results. Most of the results do not show statistical difference between the LTGA and GOMEA-SAT. GOMEA-SAT even manages to perform better in some benchmarks, while the LTGA approach performs significantly better at only one instance ("Glassy" with $s_{LT_{250}}$).

These results show that while the Linkage Tree learning generates good and comparable results to GOMEA-SAT, it fails to significantly improve upon them. Furthermore, LTGA's tree generation process is very expensive making it difficult to use on benchmarks with very large amounts of variables. GOMEA-SAT is capable of producing similar, and often better results with less overhead than LTGA.

Chapter 6

Conclusions

6.1 Summary

A new SAT Problem solving genetic algorithm was introduced, developed and tested throughout this paper. In the first stage of this project, research was conducted in order to find the correct and most optimal configuration of GOMEA applied to the satisfiability problem. This included testing the aspects of population size manipulation, evolution length, modification of the Family of Subsets structure and addition of the Forced Improvement mechanism.

Once the pure genetic system was optimized, a neighborhood search was introduced into the algorithm in order to further improve the results. With the addition of the Walksat local search algorithm, GOMEA-SAT became competitive with successful SAT Problem solvers such as GASAT or Walksat.

Due to its genetic nature, GOMEA-SAT carries a lot of overhead, and can take up a lot of processing power compared to other algorithms in order to catch up with them. This makes it an unnecessarily complicated system for solving small SAT Problems, as other simpler algorithms might accomplish it faster.

However, with time, this new GOMEA-SAT algorithm becomes more powerful, and for harder problems it becomes more efficient and often significantly better than the other algorithms it was tested against. This shows that the GOMEA-SAT based genetic approach can be very powerful and successful in solving satisfiability problems.

Not all the hypothesis throughout this paper have turned out to be true. Some experiments turned out successful - such as the addition of Forced Improvement into GOMEA. Some turned out to have little or no effect, such as the Tournament Selection. However the research goal was met. The Gene-pool Optimal Mixing Evolutionary approach enhanced with a local search algorithm resulted in a competitive SAT Problem solving algorithm - GOMEA-SAT.

6.2 Future Work

After a lot of optimization and testing, GOMEA-SAT shows a lot of promise as a successful SAT Problem solving algorithm. At its most optimal configuration it is capable of competing against successful algorithms such as Walksat or GASAT.

During the testing of this system it became clear that even though creating the Family of Subsets structure by a direct mapping from the SAT Problem formula is a very efficient approach, it is not the most optimal one. Creating the clause structure based on the Boolean formula of the SAT Problem is crucial to the performance of GOMEA-SAT, but could still be improved upon.

The LTGA approach, which uses linkage learning in order to generate the FOS structure is a possible alternative, however due to its potentially high time complexity it is not feasible for a vast number of SAT Problems, due to the large amount of variables and clauses present. The experimentation data also shows that the LTGA approach failed to significantly improve upon the GOMEA-SAT results. This does not mean however that learning of structure and dynamic subset manipulation cannot improve the results.

An addition of a lighter version of learning based on variable correlation could be very beneficial to GOMEA-SAT and produce even better results in the future. This will however require a significant amount of research. As shown in the results of the GOMEA-SAT optimization section, changes to the FOS structure can have very devastating effects on the result. This is why any learning additions and modifications to the FOS structure will have to be carefully designed, implemented and extensively tested.

Appendix A

Benchmark Overview

All benchmarks used in this research were from the official benchmark library - SATLIB - The Satisfiability Library, and were used in the SAT Competitions at some point in the past.

Below is the full summary of all benchmarks used in this paper, research and all the experiments

Full Name	glassyb-v399-s732524269.shuffled-as.sat03-1680.cnf
Abbreviation Name	Glassy
Type	Random
Number of Variables	399
Number of Clauses	1826

Full Name	hidden-k3-s2-r4-n500-01-S1373238829.shuffled-as.sat03-1035.cnf
Abbreviation Name	Hidden
Type	Random
Number of Variables	500
Number of Clauses	2000

Full Name	hgen8-n120-03-S1962183220.shuffled-as.sat03-877.cnf
Abbreviation Name	Hgen8
Type	Random
Number of Variables	120
Number of Clauses	198

Full Name	homer19.shuffled-as.sat03-430.cnf
Abbreviation Name	Homer19
Type	Industrial
Number of Variables	330
Number of Clauses	2340

Full Name	homer17.shuffled-as.sat03-430.cnf
Abbreviation Name	Homer17
Type	Industrial
Number of Variables	286
Number of Clauses	1742

Full Name	genurq4Sat.shuffled-as.sat03-1510.cnf
Abbreviation Name	Genurq4
Type	Handmade
Number of Variables	64
Number of Clauses	298

Full Name	2000009987nc.shuffled-as.sat03-1665.cnf
Abbreviation Name	2nc
Type	Handmade
Number of Variables	2756
Number of Clauses	10886

Bibliography

[1] Peter A. N. Bosman and Dirk Thierens, *Linkage Neighbors, Optimal Mixing and Forced Improvements in Genetic Algorithms*. In Proceedings of the Genetic and Evolutionary Computation Conference - GECCO-2012.

[2] Peter A. N. Bosman and Dirk Thierens, *Genetic Recombination, Distribution Sampling and Optimal Mixing with a Family of Subsets in Evolutionary Algorithms* (2011).

[3] Stephen A. Cook, *The complexity of theorem proving procedures* in Proceedings of the Third Annual Symposium on Theory of Computing, pp. 151-158 (1971)

[4] Stefan Harmeling, *Solving Satisfiability Problems with Genetic Algorithms* (March 9, 2000).

[5] John H. Holland, *Genetic Algorithms*, (2001), available at `http://www2.econ.iastate.edu/tesfatsi/holland.gaintro.htm`.

[6] Holger H. Hoos and Thomas Stutzle, *SATLIB: An Online Resource for Research on SAT*. In: I.P.Gent, H.v.Maaren, T.Walsh, editors, SAT 2000, pp.283-292, IOS Press, 2000. SATLIB is available online at `www.satlib.org`

[7] Frederic Lardeux and Frederic Saubion and Jin-kao Hao, *GASAT: A genetic local search algorithm for the satisfiability problem*, Evolutionary Computation (2006, Vol 14), pp. 223–253.

[8] Frederic Lardeux and Frederic Saubion and Jin-kao Hao, *Recombination Operators for Satisfiability Problems*, Lecture Notes in Computer Science (2004, Vol 2936/2004)

[9] Brad L. Miller and David E. Goldberg, *Genetic Algorithms, Tournament Selection, and the Effects of Noise*. In Complex Systems Vol.9 pp. 193-212 (1995).

[10] Bart Selman and Henry Kautz and Bram Cohen, *Local Search Strategies for Satisfiability Testing*. In, Dimacs Series in Descrete Mathematics and Theoretical Computer Science, pp. 521-532 (1995).

[11] Joao Marques-Silva, *Practical Applications of Boolean Satisfiability*. In, Workshop on Discrete Event Systems (WODES'08), Goteborg, Sweden, IEEE Press (2008).

[12] Dirk Thierens and Peter A. N. Bosman, *Predetermined versus Learned Linkage Models*. In Proceedings of the Genetic and Evolutionary Computation Conference - GECCO-2012.

[13] Dirk Thierens and Peter A. N. Bosman, *Optimal Mixing Evolutionary Algorithms*. In Proceedings of the Genetic and Evolutionary Computation Conference - pages 617-624, ACM Press, New York, New York (2011).