Solving the vehicle routing problem with a multi-agent *Physarum* model

David M. Versluis
david.m.versluis@gmail.com

31-3-2018

Abstract

The vehicle routing problem (VRP) is an important NP-complete combinatorial optimization problem, with many applications. This thesis presents a novel method for solving two-dimensional Euclidean VRPs based on a multi-agent model inspired by the slime mold *Physarum polycephalum*. The model consists of a large population of particles initialized around the depot, from where they spread across the environment. Each vehicle is represented by a different kind of particle. Each particle is attracted to cities and to other particles of its kind. A solution is generated by letting each population of particles cover a number of cities, in such a way that all cities are covered. The outline of each population then determines both which cities should be visited by which vehicle, as well as the tour that should be taken by that vehicle. By using the properties of the multi-agent *Physarum* material and a feedback system based on the cities covered by each population, valid VRP solutions can be generated. The model was tested with 8 different problem sets with different problem parameters, each problem set consisting of 20 different problems, and each problem being run 20 times. The problem sets varied in parameters from 2 vehicles on 10 cities with 6 capacity each, to 3 vehicles on 20 cities with 9 capacity each. Performance was variable between problems and between problem sets, from solutions on average 23.6% longer than optimal on the former to at least 52.2% longer than optimal on the latter. Performance was generally superior on problem sets with fewer cities and larger capacities, compared to those with more cities, narrower capacity requirements, or both. The mechanics of solution formation are observed and discussed. While the model is not competitive in solution quality, it is nonetheless notable for its novel method of operation, consisting of complex behaviour from simple principles. Several techniques used may be of interest for improving or creating other multi-agent *Physarum* solvers.

Contents

Part I

Background

Chapter 1

Introduction

The vehicle routing problem (VRP) is an NP-complete combinatorial optimization problem [34]. In its simplest form, the VRP consists of a number of vehicles, usually trucks, and a number of points, one of which is dubbed the depot, on a graph. The points usually represent cities. The vehicles all make a tour, which begins and ends at the depot, and must together visit all other points once. The goal is to minimize the total route length. Effectively, each vehicle is assigned a number of points, and travels like a travelling salesman between those points and the depot. The VRP has many practical uses, with applications ranging from newspaper distribution to oyster planting [15].

Many methods exist for solving the VRP [34]. These methods revolve around solving the two interdependent tasks of partitioning the cities among vehicles, and deciding what route the vehicles should take through their cities [34]. Many different exact and heuristic methods exist,usually functioning algorithmically. Bio-inspired VRP algorithms also exist, such as ant colony optimization, artificial immune systems, and particle swarm optimization algorithms [31]. It has not been previously solved by a multi-agent bio-inspired model using material computation.

This thesis presents a heuristical VRP solver based on a multi-agent model of *Physarum polycephalum*,a slime mold commonly cultivated for scientific purposes. *Physarum polycephalum* has recently become renowned for its apparently intelligent solutions to difficult problems [33]. A single *Physarum* organism created a close approximation of the railway network around Tokyo, based only on city locations (a similar but simpler network in figure 1.1). It was capable of doing this due to the unique morphology of the slime molds. In its adult stage *Physarum polycephalum* consists of a single macroscopic multinucleated cell, often irregularly shaped and measuring several centimetres across (figure 1.1) [28]. Multi-agent models of *Physarum* have been capable of creating approximate solutions for complex combinatorial optimization problems, such as the travelling salesman problem (TSP) [22].

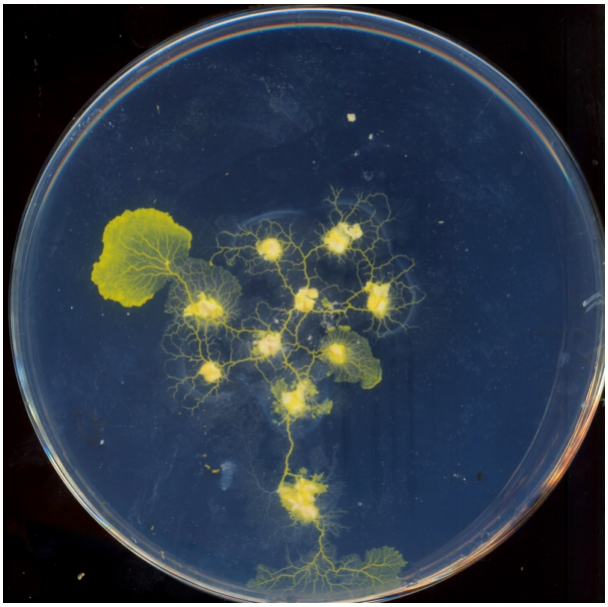


Figure 1.1: A network between oat flakes created by *Physarum polycephalum*. From [2].

The *Physarum* VRP solver presented in this paper functions by having a large population of interacting particles representing each vehicle spread from the depot across a two-dimensional environment. This environment also incorporates the city locations. Jointly, the particles compute a tour for each vehicle, leading to a VRP solution.

The structure of this thesis is as follows: chapter 2 and 3 are dedicated to the further background of *Physarum* computing and VRP computing, respectively. Chapter 4 introduces the requirements of a *Physarum* VRP solver, and the choices made in creating it. Chapter 5 consists of a full description of the *Physarum* VRP solver. Chapter 6 describes the generation of the problem sets and the solutions generated to be compared with the results of the *Physarum* VRP solver. Chapter 7 discusses the results, both in general and on several case studies. Chapter 8 discusses the outcome and relates it with other works, leading to the conclusion in chapter 9. Chapters 10, 11, and 12 contain additional figures and background information.

Chapter 2

Physarum computation

This chapter discusses some relevant background consisting of previous *Physarum* research, especially multi-agent *Physarum* models, of which the model in this thesis is a member. The simpler problem solving of the living *Physarum* and early multi-agent models will first be discussed, followed by the graph solvers and finally the TSP solver that has been developed.

2.1 Natural *Physarum*

The life stage *Physarum* is usually studied in is its adult stage, where it consists of a large multinucleated cell [28]. In this stage, it is known as a plasmodium. The plasmodium has no specialized tissue, and no centralized regulation. In nature, it is common on dead wood, feeding on the bacteria present. The plasmodium grows and extends itself until it forms a sort of network, covering multiple food sources at once. By continuously moving its internal fluid around it can both move and grow to gather nutrients and transport these nutrients around the cell. This is called the foraging or migration phase, and is the most studied and modelled phase. During this phase the plasmodium forms a continuous, advancing front, supported by many major and lesser veins [28] (upper left of figure 1.1). Should nothing be found in a particular direction, the internal fluid can flow back out of the branch entirely, leaving only a thin layer of slime (bottom of figure 1.1).

When placed on a substrate that is barren except for a few concentrated food sources, usually oats, the same pattern of movement and growth occurs, the *Physarum* remains on and between the food sources. This leads to a distributed network that is often strikingly similar to one designed by humans. *Physarum* has not only been used to recreate the Tokyo railway network - other applications have been in such varied situations as designing networks between future lunar bases and recreating ancient Roman Balkan roads [3, 12].

Physarum has even been used to solve some simple graph problems, such as creating a concave hull around a set of points [2]. In this experiment each bit of food represents

a point. However, even this solution required inoculation at the points with a substance that is both attractant and repellant at the same time, and considerable time to let the *Physarum* grow into the right shape. While growing into a network, the plasmodium itself approximates various proximity graphs [1]. Many of the more complex graph problems, such as the TSP, are beyond the reach of a *Physarum* plasmodium [22]. Only by interfacing the plasmodium with a dynamic feedback system that selectively limits the growth of the plasmodium, controlled by a recurrent neural network, have solutions for very simple TSPs of four cities been reached [4]. Turning to simulations of *Physarum* has allowed for more complex problems to be solved than with living *Physarum*.

2.2 Simple multi-agent *Physarum* models

An early *Physarum* multi-agent models, which all models discussed here are based on, was created in 2010 [18]. This model, and the further work using it, will be discussed in this chapter. The model uses a two-dimensional flat plane in which agent particles can move. The *Physarum* is modelled by a number of agent particles, each of which occupies a single location in the environment at any given time. The particles each have three sensors, all located at some distance in front of the particle and from each other (figure 2.1). These sensors are used to measure the local concentration of a local variable that is known under various names that will be called "attractant" here. It exists on the same two-dimensional plane as the particles, but its interaction is limited. Effectively, it attracts particles; at each step of the model each sensor senses how much attractant is underneath it, and the associated particle rotates to the left or right if its rightmost or leftmost sensor detects more than the front sensor. The particles also move forward each step. Because the particles will turn to (somewhat) face the largest concentration, the particles will move towards, and sometimes past, sources of attractant. The crucial step in causing this model to generate interesting behaviour is that the particles themselves are the sources of attractant. Each time a particle moves successfully, it deposits a small amount of attractant. A particles move is always successful, unless its intended location was already taken. In that case the particle chooses a new random direction to attempt to move in next time, and does not deposit attractant. After movement has occurred, the attractant diffuses from its current squares to neighbouring squares. Over time this produces a gradient.

In the environment used here, any particle or attractant that leaves the area moves to the opposite side of the area. This means all sides are essentially stitched to their opposite side, i.e. the environment forms a torus. Attractant is also affected by this, as diffusion can occur across the edge.

The behaviour of the particles from these initial settings depends, of course, on the parameter settings. The particles starting location and facing is randomized for each particle. Of particular importance are the rotation angle (RA) and the sensor angle (SA) of the particles, which are set universally and do not change. For one set of experiments

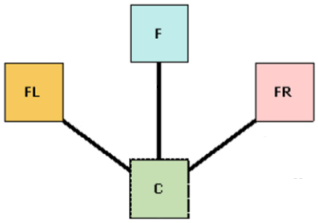


Figure 2.1: A particle in the *Physarum* model, showing the particle position C and the offset sensor positions FL, F, and FR. From [22].

RA was set at 45 and SA at 15 [18]. This means that rotation only happened in steps of 45 degrees, and that two of the sensors were offset 15 degrees to opposite sides from the front sensor. With these settings, the particles show their interesting emergent behaviour; they spontaneously develop dynamic networks (figure 2.2). The particles start at random positions, but quickly coalesce into paths of particles (figure 2.3). Such a path consists of an approximate line of attractant along which particles move in both directions. While moving along it, the particles deposit attractant, strengthening the path. These paths then form into a dynamic network that spans the environment. This behaviour is an example of stigmergy, the emergence of coherent activity through indirect coordination. The network remains in a state of flux, with paths merging together and splitting off from existing paths.

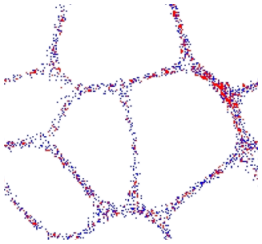


Figure 2.2: An emergent dynamic network. From [18].

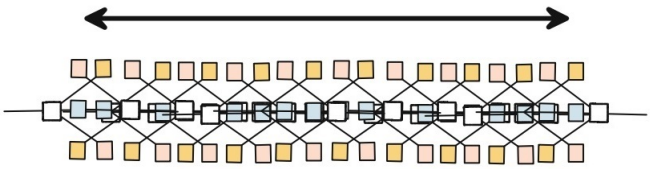


Figure 2.3: Schematic representation of a path of the particles shown in figure 2.1, as spontaneously forms from random initializations of particles. From [20].

The same permanent state of flux is observed with different parameters, as long as they are set at such values that the rightmost and leftmost sensors can become placed entirely

out of the main path stream by a single rotation [18]. This allows for new paths to form, as the particle moves away from the main path and is followed by other particles.

By setting the SA at 45, equal to the RA, networks that reach equilibrium develop [18, 19]. Paths still form and merge in the initial stages, but new paths no longer split off from those that already exist. Paths still merge to some extent until a regular pattern is formed, which is stable (figure 2.4). Although the particles themselves remain in movement, the overall pattern is in equilibrium.The exact pattern of this tiling is somewhat random, being influenced by the initial random distribution and orientation. However, it consistently approximates hexagons when tiled.

Figure 2.4: An emergent equilibrium state. From [18].

Different behaviour is observed by letting the edge of the environment eliminate attractant and particles instead of letting them through to the opposite side [18, 19]. RA and SA are maintained at 45, and the agents are again observed to coalesce into paths. The merging of paths continues until all particles are part of a single round sheet-like mass with little internal movement. Particles can only move in small areas inside of the mass. As density increases and holes in the sheet shrink and disappear, the amount of attractant within the sheet decreases. Eventually nearly all the attractant is at the edge of the sheet, as that is the only place where successful movement can still occur. Lowering SA to a lower value such as 22.5 results in some mergers, but no solid sheet-like mass, due to the higher propensity towards branching behaviour.

While these spontaneous networks show interesting structures, they can also be guided by underlying stimuli to create useful behaviors [18, 19]. The stimuli take the form of a point in the space, a "node", which emanates some amount of attractant (the amount being a parameter), simulating the bits of food used in the experiments with live *Physarum*.

Three different methods were used to examine the interaction between nodes and particles. Each will be described in its own section now.

Filamentous condensation

The first of these is the filamentous condensation method. As was done before, the particles are initialized at random locations with random headings [18]. The addition of the nodes causes the particles to form their network around the nodes. The network can still minimize

if it normally would, eventually existing only between the nodes and Steiner points (figure 2.5). Steiner points are points that are equidistant from multiple nodes, by going through these points the total network length is reduced. A network with nodes that deposit a very large amount of attractant does not have Steiner points [19]. Minimization of the network may cause it to not include one or more of the nodes.

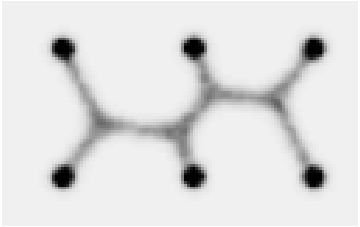


Figure 2.5: A network with Steiner points resulting from filamentous condensation. From [18], where it is called filamentous shrinkage.

Filamentous foraging

In the second method, filamentous foraging, the initial population starts at the nodes with random headings [20]. They are gradually placed there during the first few steps of the program, to prevent overcrowding. To cause the agents to engage in exploratory behaviour and leave the direct vicinity of their original node, the sensor angle is 15 at the outset. It is increased to 45 once a network forms, to cause the network to stabilize. The networks produced are similar to those created by filamentous condensation, minimizing to Steiner points depending on how strongly the nodes attract.

Plasmodial shrinkage

The third and final method, plasmodial shrinkage, uses a very large population that is initialized across the environment [20]. This will form into a sheet of particles similar to a plasmodium of *Physarum*. Each particle has been given a very small constant chance of dying each step. Over time, this causes the sheet to shrink, until only a network between the nodes remains. The network produced is similar to that obtained by the previous methods, but has some interesting intermediate states and utilizations, which are described later in this chapter.

This method is also interesting in its analogy to live *Physarum*. While with all previously used settings the particles formed into open structures similar to the foraging structures of *Physarum*, this solid mass is similar in structure to the fed plasmodium of *Physarum* that shrinks to form a more efficient network between its food sources[19].

2.3 Solving graph problems

The generating properties of a *Physarum* model have been used to generate various interesting graphs [20]. One that is of particular interest for the VRP is that of the convex hull, which can either be generated by the model, or used as a starting point for the model. The convex hull is the smallest convex polygon enclosing a set of vertices where all vertices are on the boundary or interior of the polygon [20]. A *Physarum* network approximates it when initialized in a wide circle around a set of vertices (nodes), as the first step towards a minimized network during plasmodial shrinkage (figure 2.6). It is also possible to initialize the particles spread randomly throughout the area within the convex hull, and then utilize the plasmodial shrinkage method. When this is done, several other phases can also be distinguished between the convex hull and the Steiner minimum tree that is ultimately achieved. These phases are similar to that observed during the latter stages of live *Physarum* foraging, when connections are culled selectively [1, 20]

Figure 2.6: The convex hull around a number of points, as formed by a *Physarum* model. From [20].

A population system was later introduced [20]. With such a system it is no longer necessary to determine how many particles should be used a priori, which allows the model to be more adaptable, dynamically conforming to the number and locations of nodes. The population system works by querying all particles at regular intervals about the number of particles in their neighbourhood. If this number is very low and the particle has moved successfully, the particle will create a new particle, provided there is space in the direct vicinity. If there are many particles in the neighbourhood, the particle dies instantly. There is a broad region of particle density where the particle will neither reproduce nor die.

2.4 Solving the travelling salesman problem

It had previously been noted, when the plasmodial shrinkage method was first described, that its outcomes could be related to the TSP solution over the same set of vertices [18]. This was subsequently developed further, into a method usable to solve many TSP problems. This solver utilizes a *Physarum* model and a variant of the plasmodial shrinkage method [22]. It is capable of accepting any Euclidean TSP, although a solution is not guaranteed. Each city of the TSP is represented by a node, and the plasmodium represents the traveller. The plasmodium is initialized inside the convex hull around the points, and left to shrink. Shrinking is achieved here not by a death chance for each particle, but by a population system similar to those previously discussed.

The model is stopped once all cities are at least partially uncovered by the plasmodium. At this point, there is a plasmodium that is close but not on all cities (figure 2.7). This does not constitute a legible TSP tour, the tour must be derived manually from the final model situation. Starting from the city closest to the top of the environment, this city is entered into the solution and the shape of the plasmodium is followed clockwise until another city is reached, which is then entered next into the solution. This process repeats until all cities are entered into the solution once, creating a TSP tour. There is some level of arbitrariness here, as cities can lie on a peninsula-like structure of the blob, and thus be encountered twice. When this happens, only the first encounter adds it into the solution, any subsequent encounters are ignored. This means multiple solutions can be derived from the same model output, if it were read counterclockwise instead of clockwise or if the starting city differed.

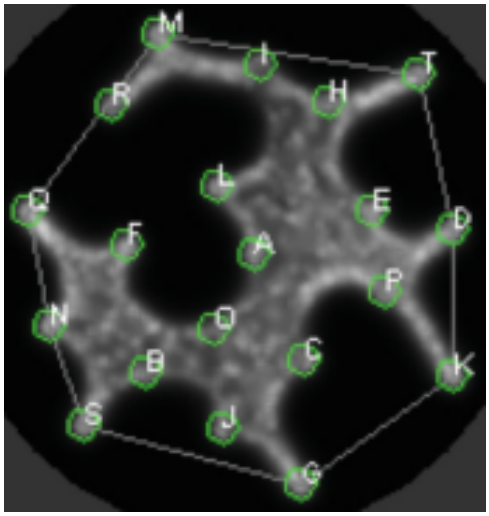


Figure 2.7: The result of the *Physarum* TSP solver, a blob that partially covers all cities. From [22].

Using 20 datasets with 20 cities each and 10 runs per dataset results were close to but longer than the shortest TSP tour possible by an average of 6.41% [22]. There was a large amount of variation between individual runs on some datasets but not with others, showing variability within the model, and also a large variation of variability.

The TSP has also been solved using a multi-agent *Physarum* model combined with a system known as dynamic reconfiguration [21]. In dynamic reconfiguration the amount of attractant produced by each city is varied according to the number of *Physarum* network connections to other cities it has. By manipulating these values to create a network where all cities have two connections, a network is generated that may be a valid TSP solution. This method did not guarantee a solution, as it was possible for two separate tours to be created. Even if it did generate a solution it was not necessarily optimal, or even remotely good. The dynamic reconfiguration necessary to sometimes generate these solutions were also more complex than that used for solving graph problems and in the TSP solver discussed above. For these reasons, the dynamic reconfiguration method of TSP solving has not been pursued further.

Chapter 3

The vehicle routing problem

This chapter briefly discusses the history of the VRP, the definition of the VRP used for the purpose of this thesis, and gives a summary of the workings and performance of other solvers on VRPs similar to those solved here.

3.1 History

While the problem of making efficient deliveries is as old as trade itself, it had not been formalized as such until relatively recently. The vehicle routing problem was first formalized in 1959 as a variation on the Travelling Salesman problem (TSP), then called the "truck dispatching problem" [10, 34]. In their formulation the salesman has to return to the "terminal point" every m of the n-1 points, where m is a divisor of n. This means that, unlike in later formalizations, all routes include an equal number of cities. The original heuristic solving method also put more emphasis on filling each vehicle to capacity, rather than minimizing tour length. The problem was later expanded to support unequal routes [7].

3.2 Problem definition

Various modern formalized versions of the vehicle routing problem (VRP) now exist, and there is no consensus on what the typical VRP should be [11]. The most studied variety is the Capacitated Vehicle Routing Problem (CVRP) [34], which is usually implied by the term VRP [34]. The definition used here is partially adapted from [24], [14] and [29]

Let $G = (V, E)$ be a two-dimensional Euclidean graph where $V = \{0, ..., n\}$ is the vertex set and $E = (v_i, v_j) :, i \neq j$ is the edge set. Vertex 0 represents the depot whereas the remaining vertices correspond to cities. A fleet of m identical vehicles is available at the depot. With every edge (v_i, v_j) is associated a distance $c_i j$. The VRP consists of designing a set of least cost vehicle routes in such a way that:

1. Every route starts and ends at the depot;
2. every city of $V \setminus \{v_0\}$ is visited exactly once by exactly one vehicle;
3. with every city is associated a nonnegative demand q_i ($q_0 = 0$), the total demand of any vehicle route may not exceed the vehicle capacity Q.

This definition is simplified from what is sometimes called the typical VRP in two ways. Firstly, it is explicitly two-dimensionally Euclidean. All distances satisfy the triangle inequality. This is a necessary simplification to solve the VRP in the two-dimensional model that will be used. The two-dimensional Euclidean VRP is still NP-hard [23]. However, triangle inequality violations do complicate the VRP [13], so their removal will decrease the difficulty somewhat. Secondly, this definition does not include a service time, which would effectively lengthen the tour with a certain value δ_i for each city and disallow any tour to exceed a preset total effective length of L. This second constraint is not always included in the "typical" VRP. In addition the variables for the experiments are set to uniform values. q_i is the same for all cities other than the depot, and all vehicles have the same capacity Q.

This problem definition and configuration were chosen to allow a multi-agent *Physarum* model to be able to attempt to solve it. Because the model runs in a two-dimensional Euclidean environment, a non-Euclidean graph can not always be properly represented. Similarly, it must be undirected, for the agents take no note of absolute direction. Other side constraints such as time windows and service times are also not used, because even this basic VRP is quite difficult to solve for the *Physarum* model. Nonetheless this version is still a full VRP, with all the requisite features.

3.3 Techniques

The great variety of (C)VRP formulations and variations make it difficult to empirically compare VRP performance [34]. In general, total route length is taken as the measure of performance. Many techniques exist to solve the VRP, which divided into two main categories: exact and heuristical.

Exact algorithms

One of the first exact algorithms for the VRP was the branch-and-bound algorithm [25], used later in this thesis to calculate exact solutions as a reference. This algorithm works by creating a tree of possible solutions, and going through the various branches systematically. Branches are pruned off of the search space by the bounds of the problem known thus far, so that not all solutions have to be checked individually.

A comprehensive overview of modern exact algorithm performance was made by P. Toth and D. Vigo [34]. The best algorithm in this overview was a Branch-Cut-and-Price

algorithm which solved problems with 199 cities (not including the depot) optimally, given enough time [30]. This uses a hybrid method that can automatically switch between two exact algorithms, a branch-and-cut and column generation method.

Heuristics

The first effective heuristic to solve the VRP as the Clarke-Wright savings algorithm [7]. This algorithm, a version of which is used later in this thesis, first makes all possible routes from the depot that visit a single city, and then merges the routes that save the most total distance and can be merged without creating an invalid solution. An invalid solution assigns a route to at least one vehicle that has more demand than that vehicle has capacity. In practical terms, the vehicle would be unable to complete the route. The process of mergings continues until no more valid merges are possible. The resulting solution is not necessarily optimal, but the algorithm is generally effective and easy to compute [34].

Further research introduced new stochastic and dynamic versions and metaheuristic search algorithms [11]. P. Toth and D. Vigo also created an overview of metaheuristics [34]. Comparing algorithms is harder here, as there are always multiple factors to consider such as quality, time spent, and flexibility. High quality solutions are possible up to approximately 500 cities [34].

There are also two particularely interesting biologically inspired meta-heuristic algorithms. One of these is especially interesting due to being inspired by *Physarum*. This *Physarum*-inspired decision maker is based on an abstract graph representation of the problem, the arcs of which are traversed by two agents representing the *Physarum* [27]. It can also be used to solve the TSP. The approach is quite different from the *Physarum* approaches mentioned previously, and from those in this thesis, because of its use of an abstracted decision graph instead of a multi-agent model with free movement over the actual problem. Using this approach, the *Physarum* decision maker solves a multi-objective variant of the VRP using a Euclidean graph of nine cities where both total route length and road traffic (i.e. route density) should be minimized. Due to these differences in both methodology and metrics, direct comparisons cannot be drawn.

A solver methodology closer to the *Physarum* approach previously discussed is ant colony optimization. It uses a large population of particles, the ants, each of which has knowledge of the graph [31]. After initialization, which can be done in a number of ways, all ants form tours, depositing an attractant as they do so. The ants choose vertices to add to their tour by a combination of its closeness and the attractant there. More efficient tours deposit more attractant on each vertex. Capacity constraints can be adhered by making each ant first return directly to the depot once the generated tour threatens to break the capacity limit. By repeating the process for a number of iterations, creating a more refined attractant map each time, a solution of good quality may be generated. Many different variations on this general method exist, and by including several other heuristics solutions can be obtained that are competitive with other solvers [6, 31].

Part II

Methods

Chapter 4

Model design

This chapter describes how the the *Physarum* TSP solving methods have been adapted and extended to solve the vehicle routing problem.

4.1 Alterations

The nature of the VRP introduces several unique complications. To solve a VRP, the set of destinations must be split into subsets, each of which is visited by a vehicle [34]. Within each subset, the shortest route that visits the depot and all the points in the subset before returning to the depot should be found. As the depot and all points have defined coordinates, this is a TSP. While *Physarum* models have previously been used to solve TSPs, this division into sets is a novel utilization. The ultimate goal is to solve both the partitioning and the TSP within those partitions at the same time, and so the VRP. The partitioning with the best TSP solutions should be found, as well as the solutions to these TSPs.

There are two more aspects that make the VRP distinct from the TSP, which need to be accounted for. The first of these is the presence of the depot in the VRP. This must be taken into account, and cannot be treated like an ordinary city if it is to be correctly integrated into each partial solution.

The second aspect and potential problem is the regulation of demand and capacity. Each city has a certain demand, and in a valid solution each vehicle can only visit a set of cities with a total demand smaller than or equal to its capacity. The capacity would be fixed and limited for each vehicle. If a vehicle would have infinite capacity in the Euclidean capacitated VRP used here, it becomes a TSP. The total route length created by that vehicle visiting the depot and all cities in the most efficient manner will be equal in length or shorter than that of any solution with multiple vehicles, because each additional vehicle would have to visit the depot separately. This would never create a shorter route. It follows that vehicles must have some defined demand limit, i.e. a capacity, in any sensible VRP.

Without some representation of this limited capacity in the simulation, invalid solutions would be generated.

To function correctly, the *Physarum* VRP solver has modifications on all levels of the *Physarum* model: the plasmodium as a whole, the cities, and the particles the plasmodia consist of. Each of the following sections dealing with the model itself will first describe the goals of the changes and then the modifications themselves.

4.2 Plasmodium

This section describes how the overall plasmodium is altered to allow a VRP solution.

Aims

The outcome of the model must have some distinction between cities belonging to each tour. This is necessary because the solution must consist of a partition into sets, each of which forms a tour when combined with an ordering. The model outcome must also give an ordering for each of the sets. It is also desirable for this differentiation to exist during the run of the model, so that temporary partial solutions can be generated and altered dynamically.

Modifications

While this is true to a certain extent for many choices in this model, the choice made here is arbitrary; because it is so basal, it is difficult to properly compare alternatives. The chosen representation is to have the multiple vehicles of the vehicle routing problem represented by multiple plasmodia in the model. Each plasmodium represents a single vehicle and has its own particles and its own attractant, which is sensed and deposited only by its own kind. The particles that make up different plasmodia are allowed to be in superposition, as are their attractants. The particles are also not initialized across the area within the convex hull, but in a much smaller area at the center of the environment. The depot (discussed in the next section) is also located there. These changes do not fulfil all the aims, this requires further changes (discussed in this chapter) and a change in how the tours themselves are derived (section 5.7).

4.3 Cities and the depot

This section describes the alterations to the nodes (cities) in the model and how the addition of the depot is handled.

Aims

The cities should be made to work well with the existence of multiple plasmodia. They should guide the shape of the plasmodia like in the *Physarum* TSP model, but with the addition of more plasmodia it is crucial for each plasmodium to be guided differently and cover a distinct set of cities. It is also desirable to have a clear and deterministic way to determine which cities belong to which set.

The depot must always influence every plasmodium, a plasmodium becoming detached from the depot may decrease the quality of the solution. While any single city may be part of any single tour, the depot must be a part of all tours.

Modifications

At the simplest level cities still perform the same function, generating attractant. The cities each have a location in the two-dimensional Euclidean plane, which remains fixed. Every cycle the cities remove a portion of all attractant around them and place new attractant, which will then diffuse across the environment.

Fulfilling the desire to have the multiple types of particles inhabit distinct areas of the environment requires some changes to the way cities produce attractant. At the start of the simulation, all cities generate all types of attractant, and so attract all particles. Because the particles are initialized in a small area around the depot, most or all cities will have no particles around them at this time. Once one kind of particle has a plurality (i.e. it is more prevalent than any single other kind) in the area around the city, it switches to producing only the attractant associated with it. A city is now said to be claimed by a certain plasmodium. As it continues to remove all attractants around it before deposition occurs, the type of attractant corresponding to the plurality will become dominant. The amount of attractant deposited does not change for the kind that has the plurality around it. Particles belonging to other kinds will be less likely to move to the city. Cities can still become claimed by another kind after this, if another kind of particle attains a plurality. Which particle kind has claimed each city is also saved as part of the final outcome, and determines directly which set each city belongs to.

The depot is considered as a special city. It always produces all kinds of attractant, regardless of the particles around it. The attractant production of the depot is much larger than that of a city. By making the depot produce far more attractant than any single city it becomes more likely that each plasmodium remains attached to it.

The radius to which cities release pheromone is expanded, as is the size of the convex hull outside of which particles are eliminated, so that particles do not leave the environment as easily, which would cause their removal. These modifications are necessary due to the difference in city claiming. In the TSP model cities are gradually uncovered, but such an uncovering is not compatible with the changes to cities described above. The retreat of the plasmodium would stop a particle kind from reliably maintaining its plurality around the

city. The removal of shrinkage also means that the normal halting condition, of all cities being uncovered, can no longer be used. This necessitates a new halting condition: this model halts when all cities are claimed.

4.4 Particle alterations and capacity

This section describes how the particles are altered in general to comply with the rest of the model, and specifically to adhere to the capacity requirements.

Aims

The plasmodium should shape itself according to its capacity. Plasmodia should not claim more cities than they can carry. By placing this control at the level of individual plasmodia, capacity can differ between plasmodia. It is important for any change to preserve the integrity of the plasmodium and preserve their stigmergy as much as possible. It is desirable to preserve the stigmergy, as this aspect is both what allows the *Physarum* material to solve problems, and what makes it an interesting solving method. It may be tempting to introduce very strong modifiers on particle behaviour itself to ensure capacity is adhered to, but such a strong interference would both greatly decrease the model simplicity and interfere with stigmergy.

Modifications

The primary measure to promote adherence to capacity is to the reproduction chance. Particle reproduction chance decreases as the number of cities claimed by its type is nearer to its capacity. By varying the decrease in population growth as the plasmodium grows, some growth can be maintained to aid in plasmodium integrity. This method of promoting adherence to capacity is a heuristic, and does not guarantee a solution that respects the capacity requirement.

While this is not the only possible way to utilize a capacity heuristic within a VRP plasmodium model, it does have several desirable characteristics. Firstly, it does not directly interfere with the behaviour of the particles, preserving their stigmergy. Because only the growth of plasmodia is limited, they can still exhibit all the usual characteristics. By being applied separately to the particles of each plasmodium it also allows for different plasmodia to have different capacities, so that a larger variety of VRPs can be solved.

Outside of the reproduction chance, the functioning of the particles themselves is kept mostly identical to that of the original, but with different parameters. Each particle still senses all particles in a pre-defined neighbourhood, and acts accordingly. If there are too many, it dies. If there are too few, it might reproduce. Kinds are not taken into account at all in this process, each particle considers all other particles as identical for this purpose. This is the only way particles of different plasmodia interact directly.

Chapter 5

Model

This chapter describes the full structure and mechanisms of the *Physarum* VRP solver. The environment will be described first, followed by the particles, the cities and the systems governing their interaction. The basic workings are identical to that of other *Physarum* models, but various aspects of particles, the cities and the population system differ, as well as the initialization, halting and reading of the tour. The default values of all parameters mentioned here are in table 5.1.

5.1 Environment

The environment consists of a two-dimensional plane measuring 139 by 139 units. Each 1x1 cell can contain any quantity of a virtual substance called attractant. Each step this substance, where present, diffuses partially to the eight neighbouring squares at a rate dictated by the "diffusion rate" parameter. Attractant is reduced by a factor of the "dampening rate" each step. Multiple types of attractant exist. They do not interact or interfere with each other in any way, but exist separately.

5.2 Particles

A number of particles exist in the environment. Each particle has a position in the environment determined by two floating point numbers. This places it on one of the 1x1 squares the environment is made of. Each particle has three sensors located at a distance to it equal to the "sensor offset" parameter. The middle sensor is appended at an arbitrary angle - the way this sensor points will be called "forward" from now. The two other sensors are appended at equal distance to either side of the middle sensor, at the angle of the "sensor angle" parameter. Each step, each sensor is used to measure the attractant concentration in the square that sensor is in. This is never the square the particle itself is in, due to the value of the sensor offset parameter used (table 5.1).

There are multiple kinds of particles. They are identical except for the attractant they sense and deposit, and the role they play around cities. Each kind of particle senses and deposits a different kind of attractant, which is unique to that kind of particle. All the particles of one kind taken together are referred to as a plasmodium here, as that is the level of organization simulated by the *Physarum* model.

If the sensor that measured the largest attractant concentration was not the forward sensor, the particle rotates towards the direction of that sensor by a number of degrees defined by the "rotation angle" variable. Regardless of the sensing and rotating, the particle now attempts to move forward by the distance of the parameter "step size". If no particle of the same kind as this particle is present in the square the particle would occupy after moving, the particle moves. After successfully moving, it deposits an amount of attractant defined by the "food deposit" variable in its new location. If the particle is unable to move, because there is already a particle of its kind in the square ahead of it, it instead chooses a new heading at random and does not move this turn. No attractant is deposited in this case.

After possibly moving, each particle has a chance, defined by the "reproduction frequency" variable (not a parameter), to sense the particles in a circle surrounding it. The radius of this circle is defined by the "neighbourhood size" parameter. If there are fewer particles of all kinds taken together in this area than the value of the "minimum particle density" parameter, the particle will attempt to divide. If there is at least one square without a particle on it within the area, division is successful and a new particle of the same kind is created on a random empty square within it.

The reproduction frequency R is calculated separately for each plasmodium during each step of the model as follows. If R is 1, reproduction is always successful, if it is 0 reproduction always fails. Each cycle, R is determined by counting the number of cities controlled by each kind of particle, and scaling it as follows.

$$R = (1 - \frac{k}{c})$$

Here R is the probability reproduction is possible for each particle per step, k is the number of cities controlled by this plasmodium, and c represents the capacity for this plasmodium. It is assumed that each city has a demand of 1.

Particles are eliminated when they stray outside of an enlarged convex hull around all cities and the depot. This area extends several squares, determined by the "city radius" variable, outside of the actual convex hull between cities.

The total number of particles is capped at a value equal to the initial number placed, the "number of particles" parameter. Should the number of particles grow to exceed this, particles will be randomly destroyed until enough have been removed. This prevents the number of particles from snowballing and slowing down the model excessively. There is also a "minimum number of particles" parameter. Should the number of particles go below

this number, new particles of all kinds in use will be created at the depot. While in normal operation the maximum number of particles is nearly always reached, the minimum is not.

5.3 Cities

Cities are represented in the environment as nodes, similar to those in previous research. Each step, each city removes an amount of attractant equal to the "food base" parameter from the area around it, as defined by the "city radius" parameter. Following this, each city senses the number of particles and their kinds within the "city radius". If there are no particles in this radius, the city deposits an amount equal to twice the "food base" of each attractant to each patch in the "city radius". If there are particles, the city only deposits the attractant corresponding to the type of particle that is most prevalent, but in the same quantity. If multiple types of particles are equally most prevalent, one is picked randomly.

The depot is represented similarly to a city, except it never changes its deposition of attractant. Each step it deposits all kinds of attractant in use around it. The amount of attractant deposited per square is equal to the "food base" multiplied by the number of cities. The amount of attractant produced by the depot does not change over time.

5.4 Parameters

All parameters are listed in table 5.1. Parameters were chosen based on some experimentation, on previously published literature related to multi-agent *Physarum* models, and by taking into consideration the computational load. For example, the initial number of particles is much lower than in the TSP solver [22]. The sensor and rotation angle were chosen to stimulate cohesion, by being equal, while allowing for some exploratory behaviour. Generally, parameters were chosen that stimulate more growth and exploration than those of the shrinking blob model, without overly diminishing cohesion. The dampening rate is lower than in the shrinking blob model, so that attractant spreads further, stimulating exploration. Conversely, The minimum particle density was set quite high, so as to strongly stimulate growth and, through growth, more exploration. Regardless, the choice of parameters is somewhat arbitrary. It was not feasible to systematically determine the influence of each parameter on the whole, especially considering the breadth of VRPs to be solved. The chosen values were found to produce workable results under a wide variety of conditions with an acceptable computational load.

Table 5.1: Parameters and their default values in the *Physarum* VRP solver

Parameter	Default value
Diffusion rate	0.5
Dampening rate	0.9
Sensor offset	2.6
Sensor angle	45
Rotation angle	45
Step size	1
Food deposit	1
Neighbourhood size	5
Minimum particle density	75
Food base	1
City radius	5
City food radius	2.5
City food strength	0.4
Number of particles	5000
Minimum number of particles	1000

5.5 Initialization

The model uses an initialization method different from those typically used in multi-agent *Physarum* models, dubbed here 'expanding blobs'. All particles, regardless of kind, are initialized at random locations in an area around the depot with a radius equal to the "city radius", and each plasmodium expands as a blob from there. This is contrary to the shrinking blob used in the *Physarum* TSP solver discussed previously [22]. Superposition of particles, even those of the same kind, is allowed during the initial placement. As particles always try to move and cannot move to a position where another particle of the same kind already exists, these superpositions will resolve over time.

5.6 Halting

The model halts at the end of the current step once all cities are claimed by a plasmodium. There is no strict enforcement of capacity included in this halting condition, which means it is possible for the model to halt in a situation where the solution is not valid, a plasmodium may have more cities than it has capacity. Variables have been altered to try to lessen this chance, but it is not guaranteed. Because the particles expand as a blob, the plasmodia obtained with this method can be read in a manner similar but not identical to that of the TSP, which will be described in the next section.

Once the model halts the claimant of each city is saved to a file alongside images of the attractant concentration per kind for each patch. Attractant functions as a useful proxy for the shape of the plasmodium [18].

5.7 Deriving a tour

Tours are derived using an algorithm based on a marching squares algorithm applied to the image of each plasmodium. This is a major divergence from earlier work, where results were derived by hand. The marching squares algorithm used is from the "contour finding" function in the SciKit 0.13.0 Python library [35]. The marching squares algorithm is a special case of the marching cubes algorithm originally designed for analyzing 3D medical scans [26, 35]. The marching squares algorithm functions by analysing the contrast between all adjacent pixels in the image, creating boundaries at the right isocline, and then interpolating between them. The value for the isocline is set beforehand, at 0.2 for these experiments.

Each image, one per plasmodium, (figure 5.1) is converted to grayscale and processed separately by the marching squares algorithm, generating a contour (figure 5.4). In some cases (such as the lower right of figure 5.2) multiple separate contours may be recognized, when an isolated piece of attractant exists somewhere in the environment. In those cases the largest contour is chosen (figure 5.5), as it should best represent the plasmodium as a whole. Each city corresponding to the plasmodium, i.e. each city claimed by the plasmodium and as such part of that tour, is matched to the single closest point on the contour. The depot is matched in the same way. The cities and depot are then put into order by starting from an arbitrary point on the edge of the contour and moving along it. Each time a point on the contour that is matched to a city or the depot is encountered, it is appended to the tour. The tour finishes when the starting point is reached again, and the final city or depot is linked to the first (figure 5.6 and 5.7). The tour length can be easily calculated from the tour and the city locations. The tour path always follows the shortest path between two locations that are consecutive in the tour, the plasmodium and contour are only used for placing the locations in a certain order. The resulting tour is compared to a solution from a traditional solver in length, which is the quality metric used, but visual comparison is also possible (figure 5.8).

The method used here is broadly similar to the manual handling used in the *Physarum* TSP solver [22], but differs in several important ways. Besides the obvious lack of human labour and bias, there is no longer a need in the model for the cities to be partially uncovered by the plasmodium, yet not uncovered entirely. Some may be on the inside, while others are on the outside. While this would create a very difficult situation for a human observer, it is trivial to calculate the distances computationally. This also leads to a reduction in ambiguity. Previously, a city that was close to two edges of the plasmodium would be entered wherever it was first encountered. This is the "peninsula problem" discussed in

chapter 2.4. This is now simply the side it is closest to. Because all calculations are performed using floating point representations, it is very unlikely that the city is actually exactly as close to two points on the contour. Only in cases where it is exactly equally distant to two points does tie-breaking occur. In these cases, the point encountered first is used, as in manual tracking. The automatic system also accommodates special case "a" in [22], in which a city should be bypassed when encountered when it should obviously belong to a later part of the tour that it is closer to. This is automatically handled correctly, by linking each city to the closest part of the contour.

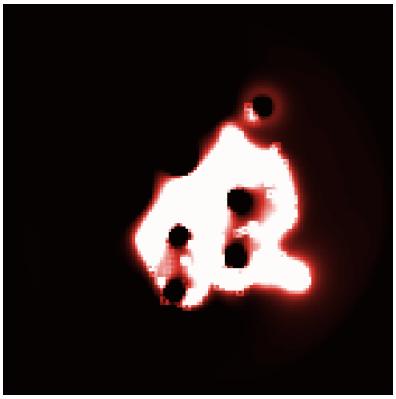


Figure 5.1: A sample plasmodium (red) as produced by the model using two kinds of plasmodium, ten cities, and six capacity.

Figure 5.2: The plasmodium (blue) complementary to that in figure 5.1.

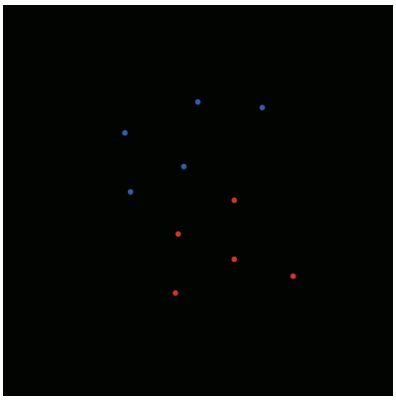


Figure 5.3: The cities and their control and location corresponding to figure 5.1 and 5.2. the depot is not indicated here. It is located in the exact center of the image.

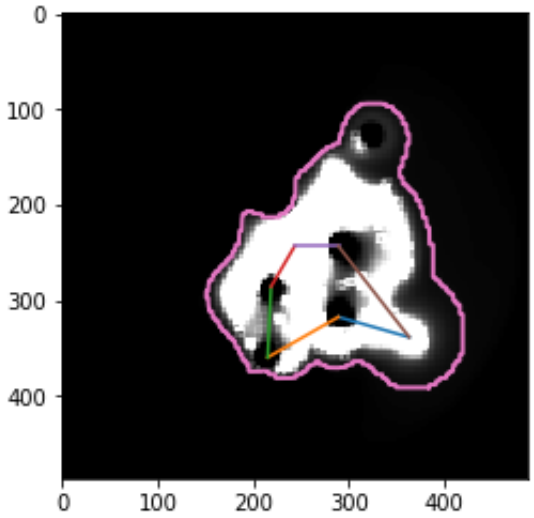


Figure 5.4: The tour and contour corresponding to the plasmodium in figure 5.1, as processed by the tour reading program. Axes are in units of pixels.

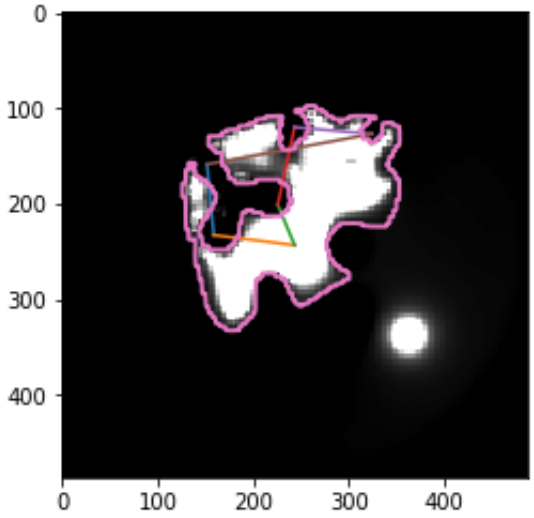


Figure 5.5: The tour and contour corresponding to the plasmodium in figure 5.2, as processed by the tour reading program. Axes are in units of pixels.

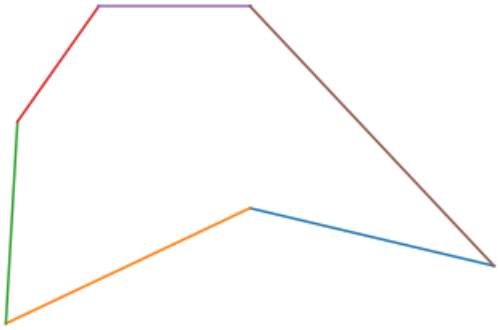


Figure 5.6: The first tour, as shown in figure 5.4.

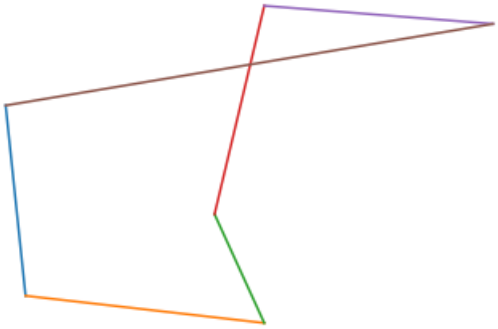


Figure 5.7: The second tour, as shown in figure 5.5.

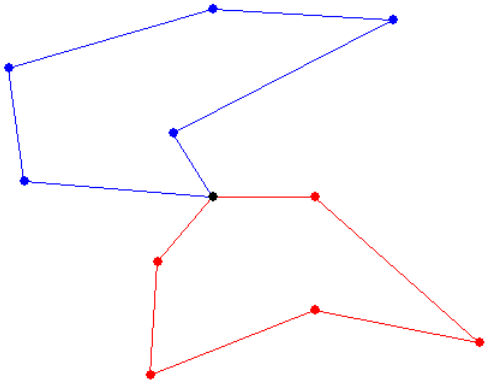


Figure 5.8: The traditional solution, as generated by an independent VRP solver [32].

Chapter 6

Problem generation

6.1 Problem parameters

A large number of problems were generated to test the *Physarum* VRP solver. Each problem consists of a city location dataset, a capacity, and a certain number of vehicles, here represented by plasmodia. Two different sets of city location datasets were used, each consisting of 20 city location datasets. The first consisted of datasets with 10 cities each, the second of datasets with 20 cities each. All datasets were randomly generated, the details of which are described in the next section. Both sets of datasets were used with both two and three vehicles (i.e. plasmodia), and with two different capacity limits, one strict and one broad (table 6.1). The strict capacity limit was set at the lowest vehicle capacity possible that has at least some margin, i.e. the sum of the individual vehicle capacities and the number of vehicles is larger than the total demand. Each city was given a demand of one, making the total demand equal to the number of cities. The broad capacity limit was set at 1.25 of the strict capacity, rounded up to the next whole number.

These parameters for problem generation were chosen to illustrate performance on both the smaller problems with broad capacity requirements where the *Physarum* VRP solver was expected to perform well, and larger problems with strict capacity requirements where it was expected to perform poorly, without failing entirely. They allow for the influence of various problem factors to be compared. There are also computational considerations: each extra plasmodium makes the model much more computationally intensive, while extra cities necessitate longer run times. Problems were limited to those where all cities and vehicles have identical characteristics (demand and capacity), to allow for easier comparison.

6.2 City location generation

The city location datasets were generated by randomly placing cities into a 109x109 area at the center of the 139x139 environment when generating 10 cities, and a 124x124 area

when generating 20 cities. The area was not increased greatly for the 20 city datasets, so that the solution could be reached with relatively few particles, ensuring computation times remained acceptable. No cities were placed close to the edge of the environment to ensure that particles and attractant will not be unduly influenced by it. Cities were also required to have a Euclidean distance of at least 10 from all other cities, regardless of the number of cities being generated. This allows cities to be distinguished for both manual and automatic tour deriving, without making the problems less challenging [22].

As all combinations of city location dataset, number of plasmodia and capacity limits form a different problem, there are a total of 8 problem sets, containing 20 VRPs each. The problems within each problem set only differ in city location dataset, not in number of cities. This is still only a small part of the entire problem space the *Physarum* VRP solver could be applied to, but it provides a variety of circumstances to demonstrate the performance and how the problem variables influence it both alone and in combination. Performance over 20 runs with each problem is used to provide information about the distribution of solutions. Performance on several sample cases is also examined.

6.3 Other solutions

To provide a comparison, all problems with 10 cities were also solved by another VRP solver using a modified version of an open source branch and bound algorithm[36], which results in optimal solutions [34]. Unfortunately, this method was not computationally feasible for the 20 city problems. For these problems a solver based on a randomized version of the Clarke-Wright savings algorithm was used [32]. Solutions for these problems are still expected to be at least close to optimal, given the relatively simple problems and the large depth and number of iterations used. The exact settings used are in table 12.1. This solver is not named, so it will be referred to as Snyder's solver, after its creator, from here on. 1000 random valid solutions were also generated for every problem. A full explanation of the generation method is in the appendix, chapter 11. These (near-)optimal and random solutions provide, respectively, an upper and lower bound of potential performance. To provide additional context, the solution for each problem from the Google OR-Tools VRP solver is also included [16]. These heuristical solutions are generated much more quickly, and are often sub-optimal and never superior to those of Snyder's solver.

Table 6.1: Problem sets used to examine the performance of the *Physarum* VRP solver

Problem set	Cities	Plasmodia	Capacity
1	10	2	6
2	10	2	8
3	10	3	4
4	10	3	5
5	20	2	11
6	20	2	14
7	20	3	7
8	20	3	9

Part III

Outcome

Chapter 7

Results

The problem sets described in the previous chapter were used to test the VRP solver. Before the overall results are discussed, several case studies will be examined in detail in the following section.

7.1 Tour construction

Figures 7.1 and 7.2 show a typical progression of states the *Physarum* VRP solver goes through while solving a problem. The problem shown here is part of problem set 1. It will be used in this section to describe some typical patterns that are apparent in the behaviour of the *Physarum* VRP solver. In addition, the final states and particle and city progression of two other runs are presented. Figure 7.7 is from problem set 5, and figure 7.10 is from problem set 7.

Initially, all particles exist in the area around the depot, and some begin to explore the area around it. The particles are attracted to the cities, and as they are claimed the other plasmodia will be repelled from it as the city removes all attractant from the area and replaces only the attractant of the kind that claimed it. These first claims set the direction of expansion of each plasmodium, as each continues to cover more area (figure 7.1). The number of particles does not typically increase greatly during this process, as the maximum number is already present at initialization. However, because plasmodia can grow during each step the total number of particles at the end of each step will often exceed the maximum (figure 7.5). Culling will then take place at the start of the next step. While the particles expand they are attracted both to cities and to the attractant deposited by other cities of their own kind. Thus, they continue to form 'blobs' of material, hence this initialization method is dubbed 'expanding blobs'.

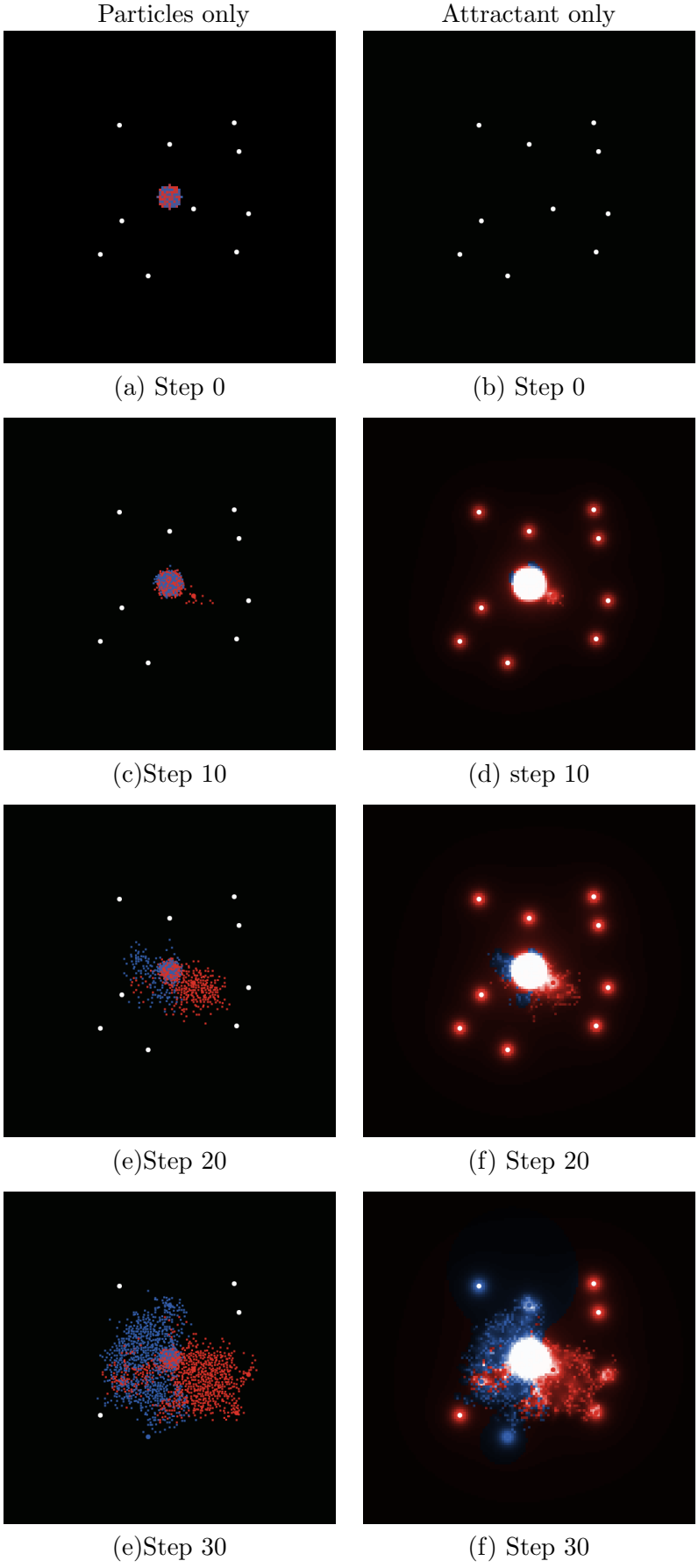
As the plasmodia continue to expand, three factors are important - their density, the number of particles they consist of, and the number of cities claimed (figure 7.6). These factors interact in various ways. Because of the maximum number of particles being en-

forced, a plasmodium can only grow sustainably at the expense of the other plasmodia. The particles culled at the start of each turn are randomly selected. With equally sized plasmodia which together take up the maximum number of particles, if one plasmodium were to grow while the rest remained constant, the plasmodium that grew would be likely to remain larger after the culling. It might then be more likely to grow even larger, as only particles can create more particles. However, reproduction is limited by the other two factors. Firstly by the density of the plasmodium, a plasmodium that is large in number but small in extent will not reproduce, as only particles in an area of low particle density can do so. In this way the plasmodia also influence each other: a dense plasmodium stops another plasmodium from expanding into it, because the area it covers already has a high particle density. Secondly, reproduction is directly lessened as more cities are claimed by a plasmodium, which is easier for more numerous plasmodia. The plasmodium with fewer cities grows more than the plasmodium with more cities, especially if the plasmodium with fewer cities already consists of more particles. The roles can then switch, with one of the plasmodia that encompassed less cities growing larger and so, with a delay, gaining more cities than the others. This leads to the newly dominant plasmodium experiencing relatively slower growth, allowing the others to catch up, and the cycle potentially repeating. These factors combine to create the pattern in figures 7.5, 7.8 and 7.11. The origins of these patterns can, at least partially, be traced back to the pattern of claimed cities in figures 7.6, 7.9, and 7.12.

After the initial exploration and claiming the plasmodia will continue to expand, being influenced by the same factors, largely along the same lines of their earlier claims and expansion. However, some exploration continues, and it is possible for this to occur despite the cities in an area being claimed by another plasmodium. An example of this is the westward expansion of plasmodium 1 (red) in figure 7.2. Eventually growth or exploration by existing particles of the plasmodia will have reached all the cities, and the model will halt (figure 7.3). The outcome shows a clear division of the set of cities between the plasmodia, and the dividing line running across the depot. The plasmodia themselves are somewhat larger than this, but this does not necessarily influence the tour. For example, the western expansion of plasmodium 1 is not nearest to any of its claimed cities, nor would it be if it were smaller, so the tour is not directly changed by it.

The factors described are also present when using different problem parameters. Because a configuration with more cities has the same maximum number of particles, the process will generally continue longer for these problems, but show the same general pattern (figure 7.7). This might cause one plasmodium to have far less particles near the end of the simulation than another, as long as it maintains more cities. Combined with the larger area that needs to be covered, the lower population can cause a plasmodium to become thinner, closer to a network than a blob. This effect is particularly strong when the population maximum is divided over three plasmodia (figure 7.10). The interactions between the plasmodia populations also become much more complex in this case (figure 7.11), although still determined by the same factors. A looser capacity will, conversely,

decrease the growth limitations. These differences in problem parameters do not pose a problem for the tour reading, as the images used for analysis have a much higher sensitivity to attractant than most images in this chapter (figure 7.3 c and d).



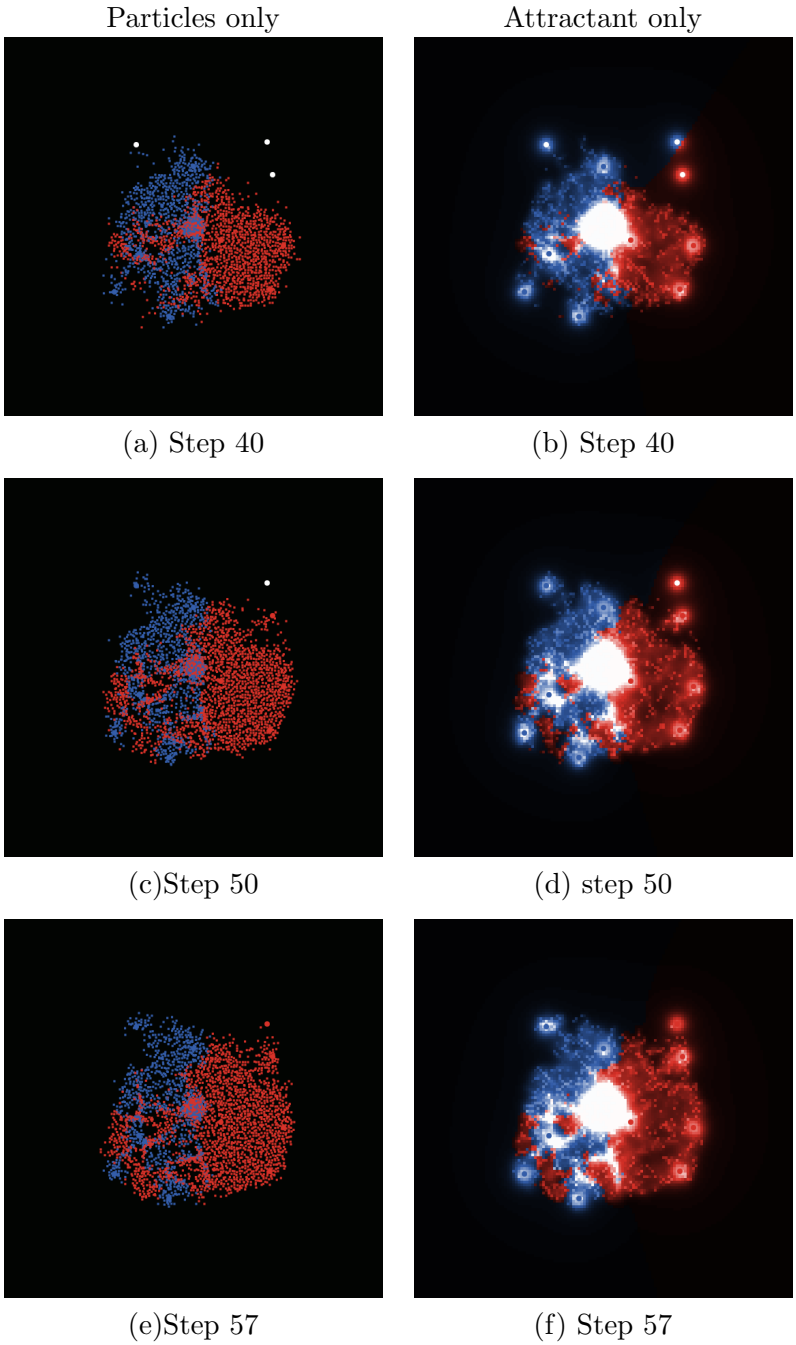
(a) Step 0 (b) Step 0

(c)Step 10 (d) step 10

(e)Step 20 (f) Step 20

(e)Step 30 (f) Step 30

Figure 7.1: A sample run from step 0 to 30 using 2 plasmodia, 6 capacity and one of the city location sets with 10 cities.



(a) Step 40 (b) Step 40

(c)Step 50 (d) step 50

(e)Step 57 (f) Step 57

Figure 7.2: A sample run from step 40 to its halting at step 57, the continuation of figure 7.1 using 2 plasmodia, 6 capacity and one of the city location sets with 10 cities.

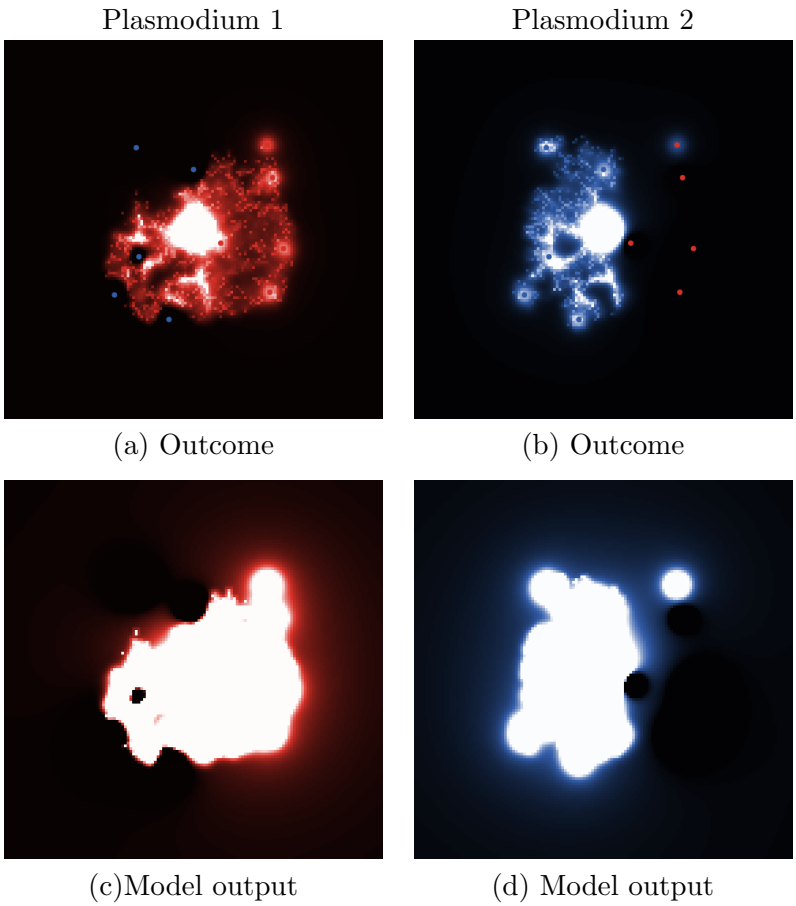


Figure 7.3: The outcome and output of a sample run, as shown in figure 7.1 and 7.2. In the output the visualization threshold has been lowered and the cities removed, to ease processing.

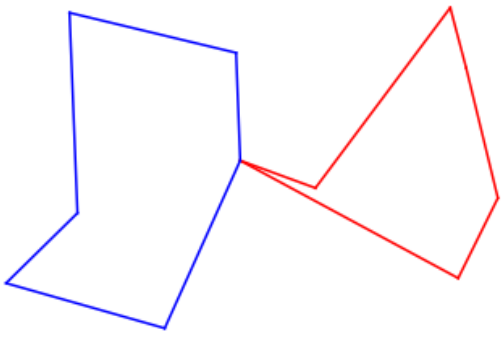


Figure 7.4: The solution derived from figure 7.3. It is of length 300, while the optimal solution is of length 285.

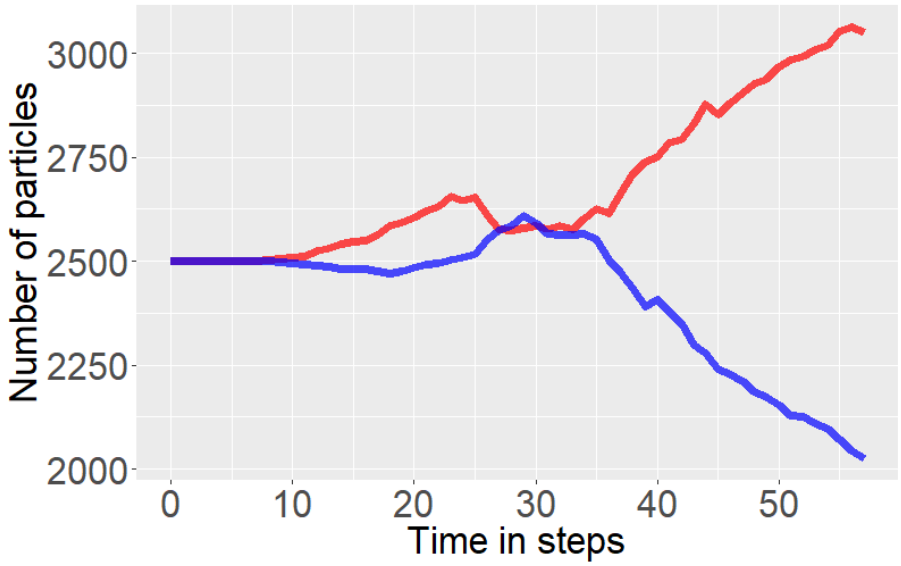


Figure 7.5: Population over time at the end of each step for the sample run from figures 7.1 and 7.2.

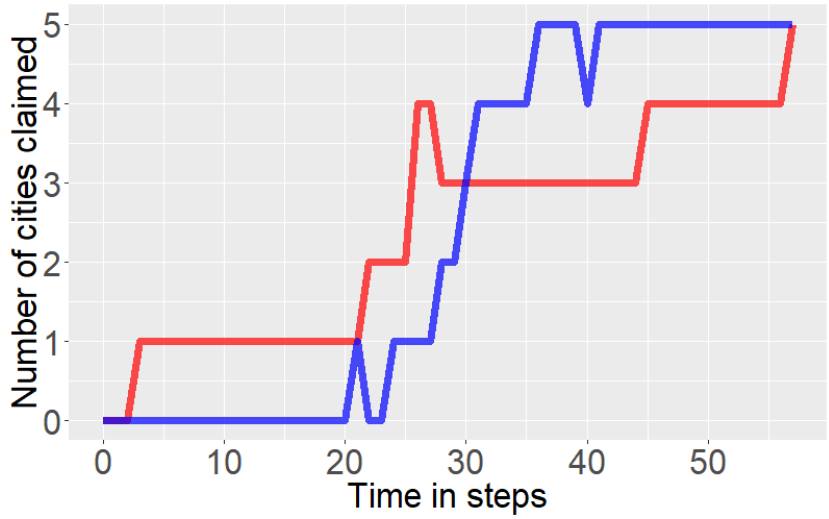
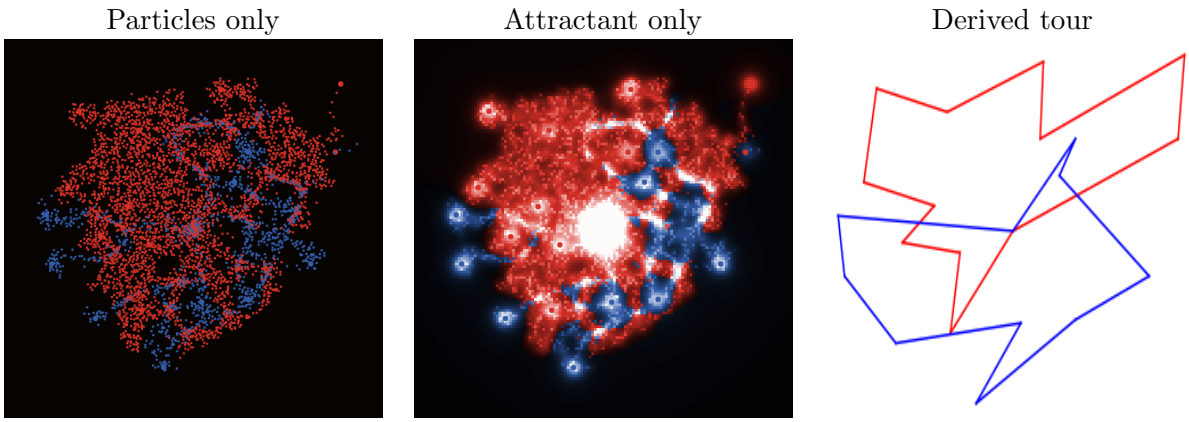


Figure 7.6: Cities claimed per plasmodium over time for the sample run from figures 7.1 and 7.2.



(a) step 130 (b) step 130 (c) final result

Figure 7.7: The final step of a sample run with 20 cities, from problem set 5. This solution is of length 678, while the solution of Snyder's solver is of length 526.

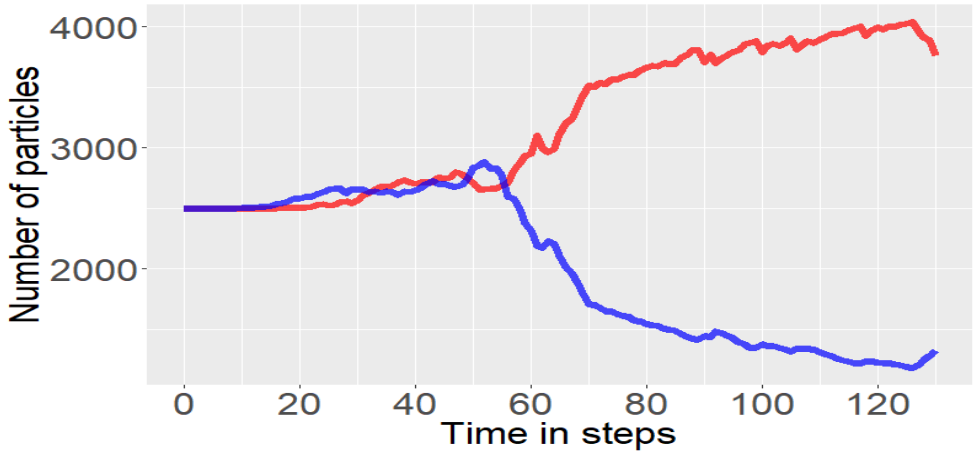


Figure 7.8: Population over time at the end of each step for the sample run from figure 7.7.

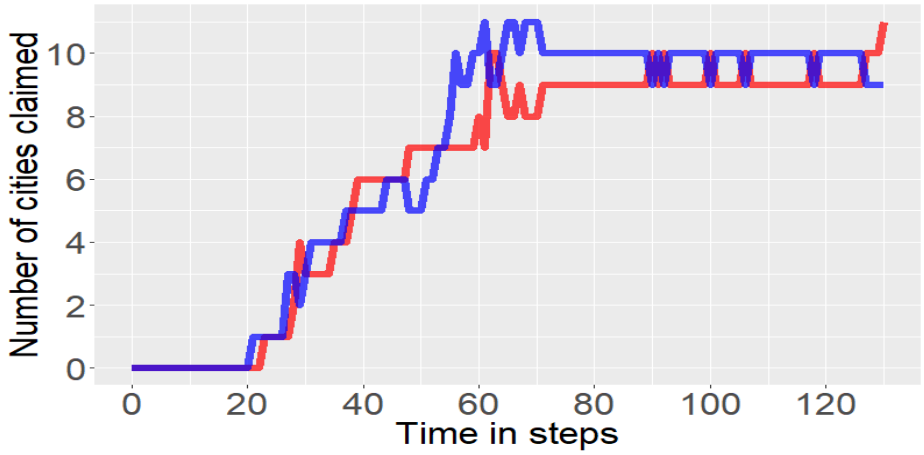
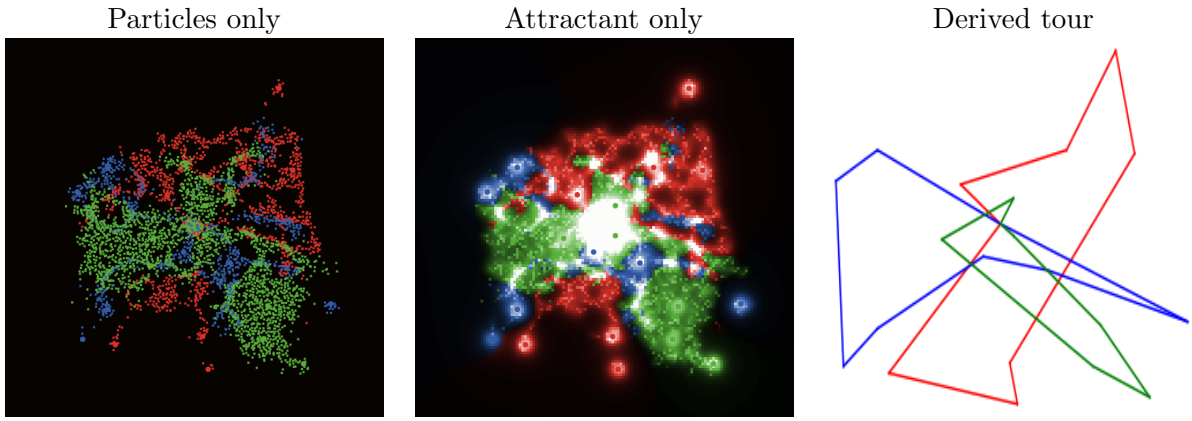


Figure 7.9: Cities claimed per plasmodium over time for the sample run from figure 7.7.



(a) step 91 (b) step 91 (c) final result

Figure 7.10: The final step of a sample run with 20 cities, from problem set 7. This solution is of length 714, while the solution of Snyder's solver is of length 510.

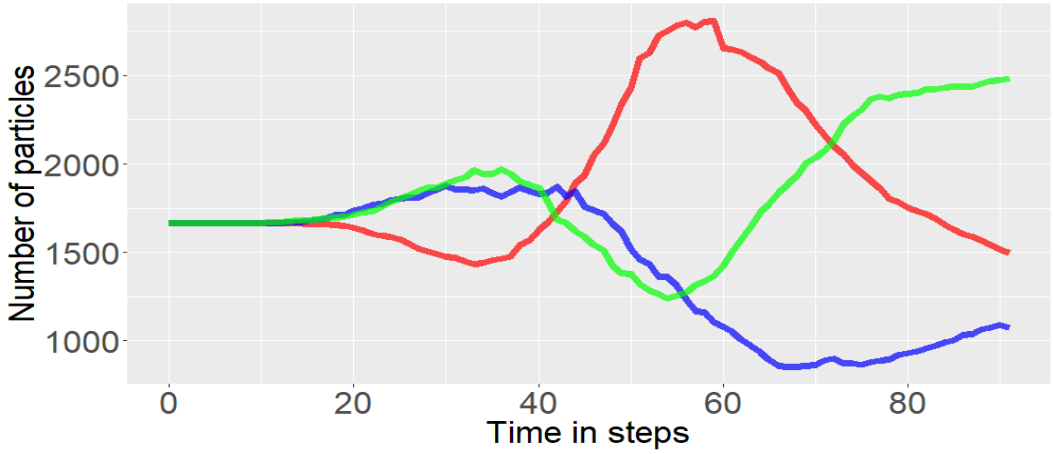


Figure 7.11: Population over time at the end of each step for the sample run from figure 7.10.

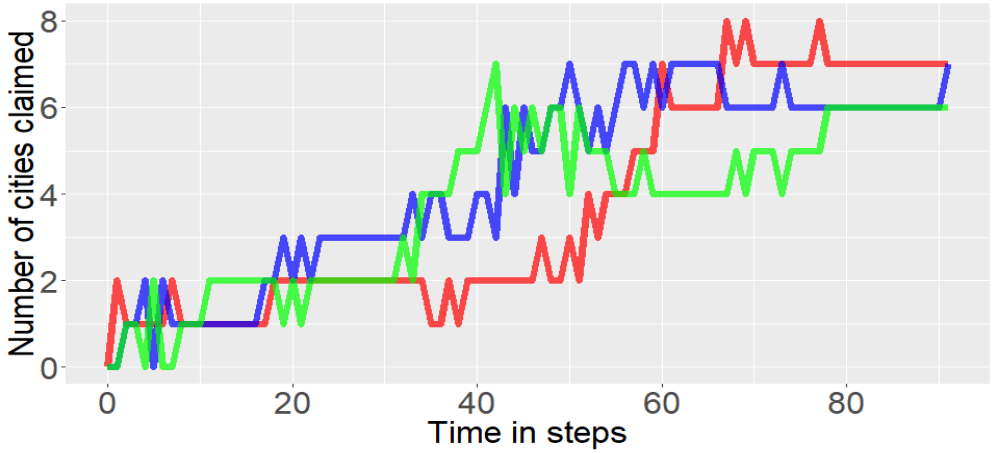


Figure 7.12: Cities claimed per plasmodium over time for the sample run from figure 7.10.

7.2 General performance

The *Physarum* VRP solver provided a large variety of solutions to the problems. Most, but not all, solutions were valid (table 7.1). Invalid solutions differ from merely suboptimal solutions in violating the capacity requirements of the problem. The bound on the tour length of invalid solutions can be lower than that of valid solutions. However, an invalid solution could never be used in practice - the vehicle would be empty before the end of its tour, and so be unable to serve the final customer(s). Because these invalid solutions are not solutions to the VRP at all, and so their inclusion would invalidate the actual results, they are excluded from those presented here.

Table 7.1: Number of invalid solutions out of the 400 solutions generated per problem set

Problem set	Invalid solutions
1	3
2	0
3	37
4	2
5	2
6	0
7	84
8	0

To allow for comparison across the various problems, the results of the *Physarum* VRP solver, the OR-Tools solver and the random solutions were normalized according to the optimal solution length of their associated problem. For the problems with 20 cities, where the optimal solution length was unavailable, Snyder's solver was used. Whenever solution length is discussed in the following chapter, it will refer to the solution length after compensation for the length of the best solution. Results were tested using one-sample t-tests to analyse deviation from the optimal/Snyder's solution, using two-sample t-tests to determine differences between sets of outcomes, and with a one way ANOVA for more general influences on the solutions. The model provided solutions that were, for each of the 160 problems, significantly shorter (i.e. better) on average than the random solutions ($p < 0.01$). Solutions for each problem were also significantly longer than both the optimal/Snyder's solution and the OR-Tools solution ($p < 0.01$). Within this range, performance varied greatly between problem sets. On average, problems with fewer cities were solved better than with more cities ($p<0.01$), solutions for problems with tighter capacity requirements were better than solutions for problems with broader capacity requirements ($p<0.01$), and problems with two vehicles were solved better than problems with three vehicles ($p<0.01$).

The results on all problem sets except for 5 were also significantly influenced by the

variation between city locations, even after compensating for the varying optimal/Snyder's tour length (one way ANOVA, $p < 0.01$).

The following section is split up by problem set. The results from the various problem sets based on the problem sets with 10 cities will be discussed first, followed by those based on problem sets with 20 cities.

Ten cities

Performance within each problem set varied greatly (figures 7.13 and 7.14). On average, solutions involving the city location datasets with 10 cities were 30.8% longer than the optimal solution. Detailed results subdivided per problem can be found in the appendix.

The means of the random sets are all far higher than any of the *Physarum* VRP solver solution sets, averaging 56% to 71% longer solutions. One set of random solutions, that of problem set 3, includes a single solution of equal length to the optimal solution.

For the first problem set (two plasmodia, strict capacity) the optimal solution was found at least once within the 20 runs for 6 problems. The best solution has a length within 5% of the optimal solution in another 9 problems. In all cases, the best solution was within 10% of the optimal solution. On average, tour length was 23.6% longer than the optimal solution.

In the second problem set, differing from the first only in the capacity of each plasmodium being 8 instead of 6, solver performance is not notably different. The number of problems solved optimally and within 5% across the problems is identical. The total tour length is, on average, 25.0% longer than that of the optimal solutions. There is no significant difference between the lengths of tours in the two conditions (two sample t-test, $p = 0.612$).

The third problem set is the first that uses three plasmodia. Because of the strict capacity requirements all three are required in each problem. Performance was significantly lower on these problems overall than on the first two sets ($p<0.01$), with a mean tour length 33.2% longer than the optimal solutions. Only one problem was solved optimally. The combination of three plasmodia and strict capacity requirements also resulted in 37 of the 400 generated solutions being invalid, more than for any other problem set using this set city location datasets.

Not all plasmodia are necessary in the fourth problem set, which has three plasmodia with looser capacity requirements. The optimal solution for 19 of the 20 problems uses only two plasmodia. Performance here was inferior to that of the third set ($p<0.01$), and so also to the first two sets, with mean tour lengths 41.9% longer than optimal. No optimal solutions were found.

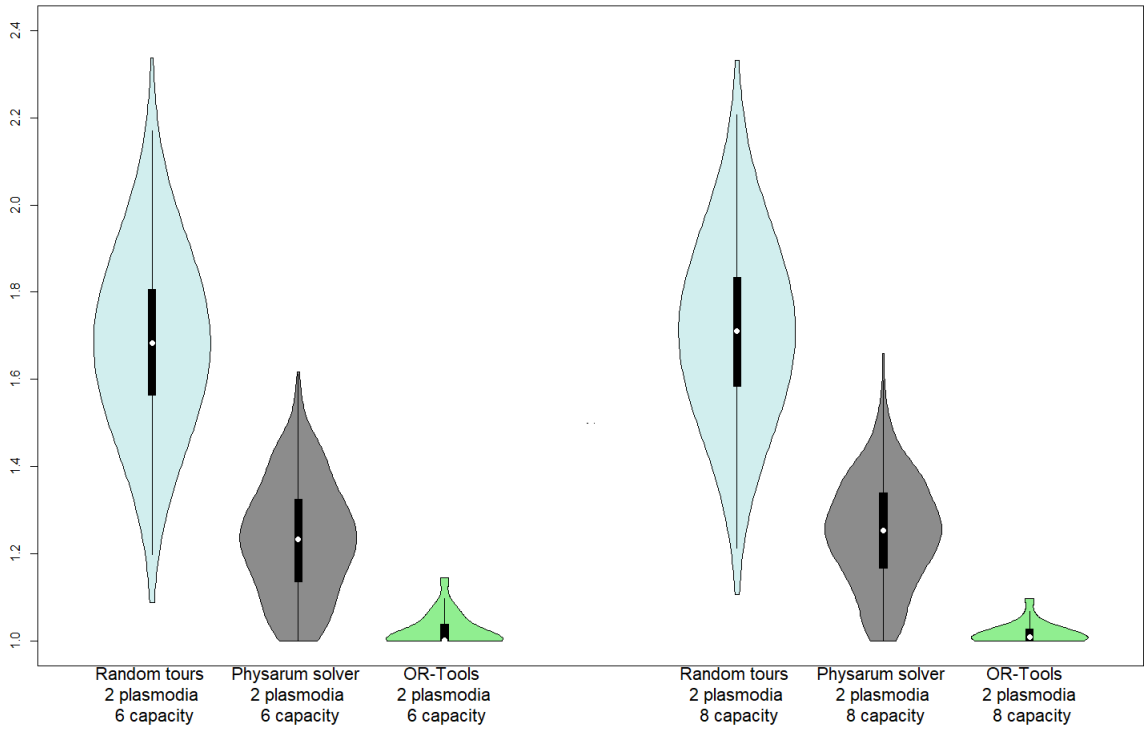


Figure 7.13: Tour length of the *Physarum* VRP solver solutions as proportion of optimal tour length for problem sets 1 and 2 and accompanying random data and OR-tools solutions. 400 attempted *Physarum* solutions per problem set, but invalid solutions excluded (table 7.1). 20000 random valid solutions per problem set. One OR-tools solution per problem. Width indicates data density, scaled separately for each shape.

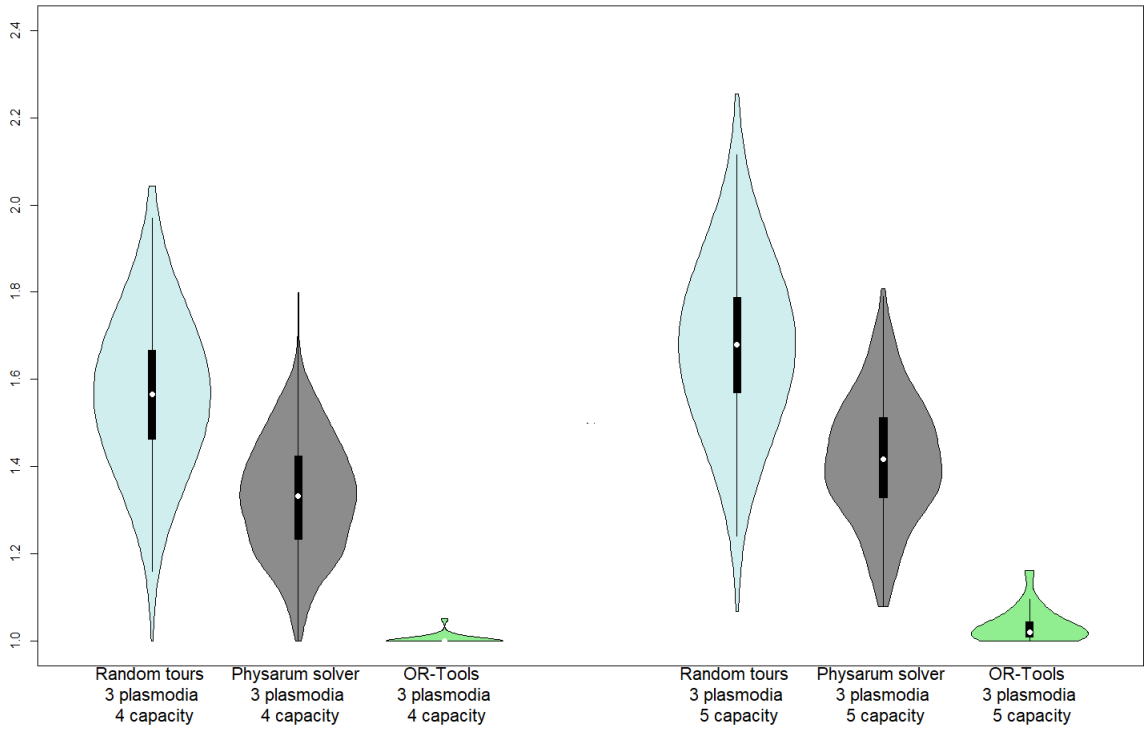


Figure 7.14: Tour length of the *Physarum* VRP solver solutions as proportion of optimal tour length for problem sets 3 and 4 and accompanying random data and OR-tools solutions. 400 attempted *Physarum* solutions per problem set, but invalid solutions excluded (table 7.1). 20000 random valid solutions per problem set. One OR-tools solution per problem. Width indicates data density, scaled separately for each shape.

Twenty cities

The following problem sets use the 20 city location datasets. As with the 10 city location sets, results subdivided per problem within each problem set can be found in the appendix. The 20 city datasets have a higher density of cities; they have twice as many cities in an area less than a third larger. None of the problems were solved as well as they were by Snyder's solver, and so also not optimally, in any of the runs using these problem sets. As in the 10 city location sets, there is considerable difference between problem sets (figures 7.15 and 7.16), and there is a significant difference ($p < 0.01$) between each set of random solutions and the associated *Physarum* VRP solver solutions. On average, solutions involving the city location datasets with 20 cities were 42.0% longer than the optimal solution, a significantly larger difference than with the 10 city location sets ($p<0.01$). Performance on each problem set was also inferior to that of the equivalent 10 city problem set ($p<0.01$).

The random sets have much longer relative solutions compared to the random sets based on the 10 city problem sets, averaging lengths 119% to 151% longer than Snyder's solutions. There were no solutions among them equal to or better than the Snyder solutions.

The fifth and six problem sets, with two plasmodia and both types of capacity, do not perform significantly differently ($p=0.41$). They have tour lengths respectively 34.2% and 33.5% longer than Snyder's solutions. Problem set 5 is notable for being the only problem set whose results are not significantly influenced by the city location sets used ($p = 0.1$).

The seventh and eight problem sets, with three plasmodia, resulted in significantly inferior performance even compared to the other 20 city problem sets ($p < 0.01$). The seventh set is on average 49.5% longer than Snyder's solutions, the eight is significantly inferior to the seventh ($p < 0.01$), being 52.2% longer. The seventh problem set is notable for having by far the highest number of invalid solutions: 84. This makes up more than a fifth of the total generated solutions for this problem set. However, there were still at least 13 valid solutions for each problem.

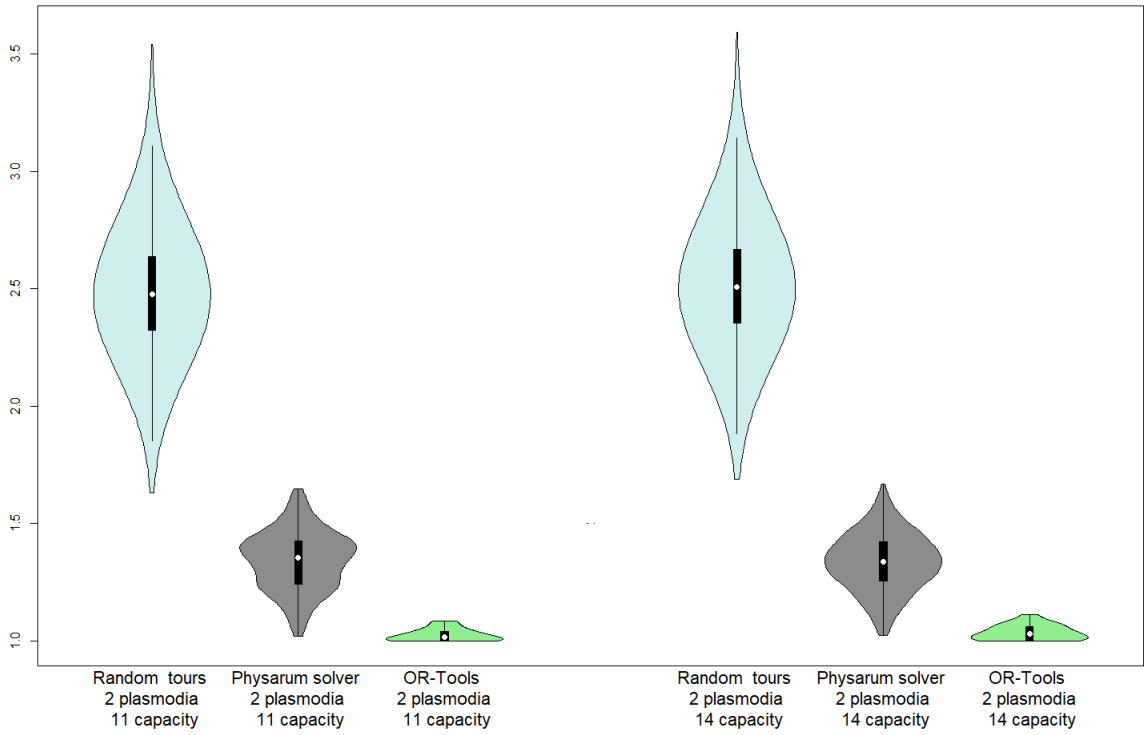


Figure 7.15: Tour length of the *Physarum* VRP solver solutions as proportion of Snyder's solver tour length for problem sets 5 and 6 and accompanying random data and OR-tools solutions. 400 attempted *Physarum* solutions per problem set, but invalid solutions excluded (table 7.1). 20000 random valid solutions per problem set. One OR-tools solution per problem. Width indicates data density, scaled separately for each shape.

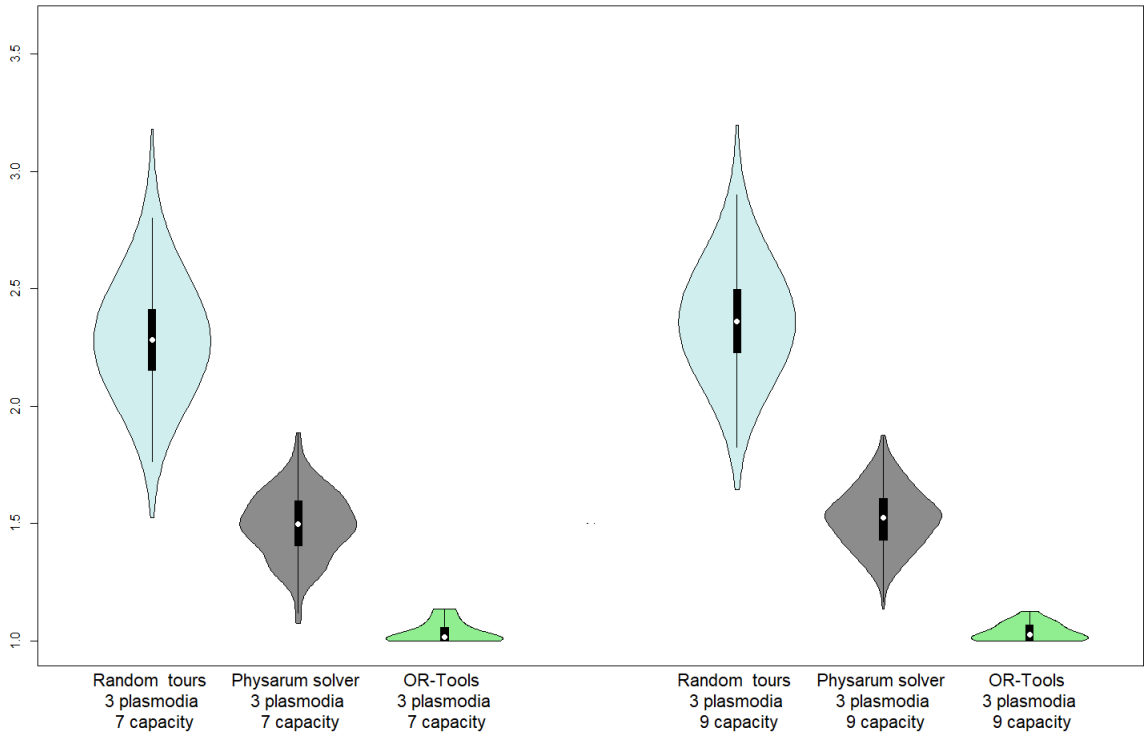


Figure 7.16: Tour length of the *Physarum* VRP solver solutions as proportion of Snyder's solver tour length for problem sets 7 and 8 and accompanying random data and OR-tools solutions. 400 attempted *Physarum* solutions per problem set, but invalid solutions excluded (table 7.1). 20000 random valid solutions per problem set. One OR-tools solution per problem. Width indicates data density, scaled separately for each shape.

Chapter 8

Discussion

This chapter first discusses the results of the solver in more depth, both in performance and in how it handles the VRP. It continues with a discussion of the limitations and of related work, especially the *Physarum* TSP solver and possible avenues of expansion.

8.1 Analysis

Performance

The *Physarum* VRP solver provided VRP solutions that were not competitive with other solvers in solution quality or computational load. However, it should be emphasised that competing with typical VRP algorithms was not the goal of the solver. Instead, it was to utilise the multi-agent *Physarum* material so that it would solve the problem at all. This goal has certainly been reached. The mere existence of performance is interesting, considering the minimal nature of the model. The solutions generated are not altered or improved in any way over what the particles themselves generate. Taking, for example, only the best of 20 runs of the model would make its solution quality similar to the OR-Tools solver on problem set 1. Similarly, simple heuristic improvements to the solution would improve the results. For example, the tour in figure 5.5 would be made identical to the optimal solution by swapping the cities where the route now crosses itself, a common heuristic improvement to TSP tours.

While overall the performance on each set is clearly influenced by the three problem variables altered, this is not necessarily the case when looking at individual city location sets. The differences between individual city location sets are also a significant factor on all but one problem set. Unfortunately, it is hard to quantify what facets of the VRP problems make them particularly hard or easy for this solver. Some indication of the variety in performance is in the figures in the appendix. On certain problems, average performance over the 20 runs is better on a problem set with relatively bad solutions overall than on

a problem set with better solutions. Nonetheless, the general pattern is clear, and also clearly significant. It is remarkable that the results of problem set 5 in particular are not significantly influenced by the differences between city location sets, especially considering that the effect is very significant in all other problem sets. It should be noted that there is still a trend towards influence by the city sets.

Not only the performance in minimizing tour length, but also the number of invalid solutions per problem set varies greatly. The failure to reach a valid solution does not seem to be caused by any individual city location sets being difficult to solve with these problem parameters, as all of the problems presented were still solved in the majority of runs. Rather, the large number of invalid solutions would be caused by the problem parameters as such. Of course, the city location sets do not represent the full possible range of city location sets, as there is little variation in density. Yet even among problem sets with the same city location set, some generate no invalid solutions, while others generate many. Problem set 7 is particularly notable for having far more invalid solutions than any other problem set. The generation of these invalid solutions is certainly to the detriment of the solver, but the solutions that are generated are not so different in quality from those in the closely related problem set 8. Using problem set 8 the solver generated far fewer invalid solutions, because of the wider capacity allowance, but performance was actually inferior to that of problem set 7. This indicates that generating more valid solutions and generating better valid solutions are not necessarily related. These two lines of improvement would have to be addressed separately in a future solver.

The particularly poor performance on generating correct solutions for problem set 7 might be partially explained by its very tight capacity requirements. The total capacity of all plasmodia is 21, while 20 is needed. This leaves little margin. Margin cannot be the only factor though, as problem set 3 has a much larger margin in relative terms of 12 capacity for 10 cities, yet it has far more invalid solutions than problem set 5, which has 22 capacity for 20 cities. The shared factor among the two problem sets with many failures is that they have both narrow capacity requirements and three plasmodia. A different handling of these problem sets, or problem sets with narrow capacity requirements in general, may be useful to reduce the number of invalid solutions.

Problem handling

The VRP is a particularly interesting problem for *Physarum* to solve because it decomposes into a problem that has previously been solved by a multi-agent *Physarum* model, and one that has not. On the one hand the TSP needs to be solved for each partition generated as a VRP solution, but on the other the partitions themselves must be generated. It is important for any solver to use the information from these parts of the problem together to come to an optimal solution. A choice of partition must be informed by (an approximation of) the length of the TSP solution of that partition. It is now clear that the stigmergic behaviour of the *Physarum* particles is suited to this task. However, it is the unique way in

which the solver handles the problem, rather than its performance, that makes it notable.

A particularly interesting aspect of the approach is the minimal amount of information particles have; particles in the model can sense only their neighbourhood. While an algorithmic solver would have access to the location of each city, the particles can only sense them indirectly, by their attractant. Nearly all other senses the particles have are also entirely local. There is only a single global factor that each particle knows, the number of cities the plasmodium it belongs to controls. This piece of information is used to manage the population, with the goal of preventing capacity violations. Effectively, each plasmodium becomes self-limiting. By slowing growth as it nears the capacity limit, and stopping it entirely once it is reached, plasmodia cause their own negative feedback. Because of the particle limit a plasmodium that grows relatively less than the others will shrink. Should the plasmodium lose some of its cities, the negative feedback stops. This simple negative feedback system works remarkably well at maintaining capacity adherence under most conditions, although as mentioned previously not under all conditions.

The solver is only influenced by one other global factor: the population limit. However, this factor is not directly sensed by the particles, in the way the number of claimed cities is. The particle limit merely culls randomly to maintain a number of particles that is low enough to compute efficiently. While an important influence on the model, it only changes the situation indirectly, as the mere act of randomly removing a small portion of particles does not greatly change the situation as a whole. It is the new growth after the culling that actually has the potential to change the relative size of the plasmodia. For these reasons it is not so undesirable as the information about plasmodium control is. Some limit on runaway population growth would be desirable in all situations, to maintain computational feasibility, but this does not necessarily have to be imposed on a global level.

Modulating the plasmodial populations by restricting reproduction may seem counter-intuitive when considering that the total population remains constant. It should be noted that the reproduction is itself a form of exploration. Because new particles are only created in areas of low density, those areas will be explored more than they normally would. A plasmodium with many cities will explore less by its lessened growth, while the attractant created by the cities causes the existing network around them to be maintained by the remaining particles. Because particles are randomly removed from all plasmodia at the end of each turn, while growth will be present at the edges, a growing plasmodium will become larger even if it does not increase in number of particles.

8.2 Limitations

Besides the performance of the model, which has already been discussed, several other limitations should also be mentioned. As was described previously, particles have some knowledge of the system as a whole, by being directly influenced by the number of cities their plasmodium controls. The model would be more interesting if the particles were truly

dependant on the information from the local area. Some other way of guiding city control would have to be found. One alternative is to keep the population of each kind of particle strictly equal, by selectively culling or spawning. This would cause roughly equal areas of control in the environment. However, this would not guarantee the capacity requirements are kept in all situations, because the cities are not necessarily spread evenly over the environment. City density does not vary greatly in the problems presented here, but there is no such requirement within the VRP problem space.

The problem space used for this thesis is also smaller than the total VRP problem space in several other ways. Firstly, all cities have identical demand in the problems used. While the solver could handle cities with differing demand by appropriately changing the reproduction rate, having cities with more than a single unit of demand could still increase the number of invalid solutions. Such a city could be added last to an expanding tour, instantly bringing it over its capacity limit. Secondly, the city location sets also do not differ as much as they could have. For example, a difference in order of magnitude would not constitute a different problem for most solvers, but without pre-modification it would in this solver. Due to the fixed size and movement distance of the particles and attractant, and the limited number of particles, absolute rather than relative distances are used.

To increase solution quality, it would be desirable for the plasmodia to remain a single unit. In the current model it is common for a plasmodium to consist of a number of cities close together and one or two cities some distance away. However, this behaviour would have to be restricted without overly limiting the dynamic exploration behaviour, which is crucial for the solver to function at all.

The parameters used for this model were chosen somewhat arbitrarily. While based on available knowledge, a systematic analysis of the parameters and their interactions could lead to a model that is much better at solving the VRP. Altering parameters such as population size based on the parameters of the problem might also help alleviate some of the difficulties, such as a large number of invalid solutions on certain problem sets.

8.3 Related work

This section discusses several avenues of related research. The primary point of comparison for this work is the multi-agent *Physarum* TSP solver [22]. There are also parallels with other solvers and biological systems, and possibilities in expanding the *Physarum* VRP solver to related problems.

Multi-agent *Physarum*

Performance is inferior to the *Physarum* TSP solver in the shared performance metric, solution length relative to optimal solution length. Even on problem set 5, the simplest problem set comparable in number of cities to the single problem set used by the TSP solver, the TSP solver found total tour lengths 6.41% longer than optimal, while the VRP

solver solutions were at least 34.2% longer than optimal. However, the VRP is a more difficult problem than the TSP - exact solutions of problems up to around 200 cities have been computed, while TSPs of tens of thousands of cities can be solved exactly [8, 34].

The TSP solver also has a far lower variation, with solution lengths ranging only between 0.45% and 20.13% longer than optimal, while the *Physarum* VRP solver solutions are between 0 and 85% longer. This is partially explained by the increased number of runs and problems, the TSP solver having only had 200 runs and 20 problems in total. But even on a small subset of the VRP data, such as only over the 20 runs for the first problem in the first problem set, the solutions vary between 1.2% and 30.9% longer than optimal. This is true for many others as well. This might be explained by the higher complexity of the VRP, as not only does each city have a place in a tour, there are also multiple tours.

The methods used in the TSP solver are, at the most basic level, the same as those used in the VRP solver. However, there are many important differences, as described in chapters 4 and 5. There are too many differences to enumerate them all exactly, so this section will focus on two interesting additions.

One difference in model methods that is of particular interest is the maximum particle density. This is present in the TSP model but abandoned entirely when using the expanding blobs model. While in the shrinking blob a minimum particle density was still worthwhile to prevent holes forming, a maximum particle density in an expanding blob serves no clear purpose. Overcrowding is already prevented by prohibiting particles of the same kind from being in superposition. This marks a decrease in particle sensing compared to the TSP, in contrast to the addition of city claim sensing compared to the TSP model. It is an important consideration if the expanding blobs model were to be used in a TSP solver.

The expanding blobs initialization method is also interesting in its similarity to one of the methods used to create networks in an earlier *Physarum* multi-agent model, the filamentous foraging method [20]. The crucial difference here is that instead of being initialised at all nodes, particles are only initialized at a single node, the depot. They are also all initialized simultaneously, not gradually. This, together with the different parameters used, causes the foraging to occur not in filaments, but as an expanding front. This expanding front covers a much larger are than the filamentous exploration.

An important methodological addition is the automatic reading of the tours. While for the TSP solvers tours had to be read manually, both taking time and introducing the possibility of bias, this can now be done entirely automatically. The methods used are generic enough to also be used with a current or future *Physarum* TSP solver, regardless of whether it uses the original shrinking blob method or another method of blob development.

Other solvers

Direct comparisons with other *Physarum* solvers are difficult, due to the radically different nature of operation. These other solvers, such as that mentioned in the introduction, do not rely on a direct respresentation of a simulated *Physarum*, as the solver presented here does.

The other VRP *Physarum* solver in particular uses an abstracted graph representation, transforming the process of solving a VRP into a graph of discrete decisions, and using a pipeline-based *Physarum* model on this graph [27]. This is quite dissimilar to the approach used in this thesis, where the representation is less abstract than typical. It also uses a different performance metric than tour length, valuing its solutions by the combination of tour length and road traffic, both of which should be minimized.

A more useful comparison can be made with the ant colony solvers, which were previously discussed in chapter 3.3. Similarly to the *Physarum* solver, particles generally start at the depot when solving the VRP [6]. A crucial difference is that the ant colony solver is entirely graph based. The particles (ants) can only move directly from city to city, and each particle knows how far away each location is. Each particle represents a vehicle, instead of a group of particles representing one. Because the ant colony system has many iterations before generating a final solution, the final tour of a vehicle will still have been influenced by many particles, but each particle will have had its own tour. There is still exploration, to determine the tour itself, but it is pre-informed at a level that the *Physarum* exploration is not. Knowledge of the capacity constraint is also absolute, with each particle knowing exactly when its capacity constraints would be violated by adding a certain city to its tour. Ant colony solvers in practice, where they are combined with simple heuristics, generate solutions of better quality then the *Physarum* VRP solver, being competitive with commonly used solvers [5, 6].

One particularly interesting aspect of the ant colony system may be used to inform a better *Physarum* VRP solver. Because the ant colony system is based on a population of solutions, it forms a complex attractant map over time [6]. While it is true that the *Physarum* has some time to alter its solutions while the tours are being made, once a possibly valid solution has been generated, i.e. all cities are claimed by a plasmodium, the model always halts. In an ant colony system this would merely be the first iteration, after which it is reinitialized with the information gained, in the form of an attractant map. Implementing such a system of pre-ingrained attractant for the *Physarum* solver may lead to much better results, at the cost of increased computational time.

Alternate vehicle routing problems

The VRP definition used in this thesis is the most basic VRP, many other VRPs exist with more complicated requirements [34]. While some of these, such as limited time windows for each delivery, would be infeasible for the current *Physarum* solver to be adapted to without radical changes, there are some variations that a *Physarum* solver may lend itself particularly well to. Two will be discussed, multiple depot vehicle routing and dynamic vehicle routing.

In a multiple depot vehicle routing problem (MDVRP) each vehicle has a specific depot assigned to it [34]. Vehicles may share a depot, or they may not, depending on the specifics of the problem. This could be modelled relatively easily in the *Physarum* solver, by creating

depots that do not generate all types of attractant, but only those of the vehicles that are housed there. The particles of each plasmodium would be initialized around their respective depot. It is expected that solutions would be generated in much the same way, and shaped by the same factors, as in the single depot VRP. Solution quality may be lowered by the reduced competition between vehicles in certain problems. If a depot has many cities but only a single vehicle associated with it, then the corresponding plasmodium could be expected to explore and claim all cities in the vicinity, even if that would violate its capacity. Capacity may have to be enforced more strongly than usual.

In a dynamic vehicle routing problem (DVRP) there is only limited a priori information about the specifics of the problem [34]. New delivery destinations may be revealed over time, as routing is in progress. In a real world application, the vehicle will have already left the depot before all its destinations are known. Because the *Physarum* solver already functions without a priori information of the problem set, it would seem to be a good fit to solving these problems. However, the method of introducing new destinations into the *Physarum* model should be chosen carefully. If cities are simply entered into the environment as their information is revealed, it would be quite easy for a large plasmodium to cover many more cities than it has capacity for. Some method would have to be devised to somewhat clear the area first, so that capacity constraints will influence which plasmodium adds the new city to its tour.

Biological systems

There is an interesting parallel in the behaviour of the VRP solver: the expanding blobs show a resemblance to the natural growth of a *Physarum* colony, foraging in multiple directions [28]. The life of a *Physarum* colony also starts from a single point, from where it explores its surroundings. Expansions that find food sources become larger, while dead ends stop expanding or die off (figure 1.1). Eventually the network expands to cover multiple food sources, and after some time halts expansion to sporulate. While the model does not sporulate, of course, the rest of the development is quite similar. The major divergence here is the presence of multiple simultaneous plasmodia. Actual *Physarum* plasmodia would merge or eliminate each other, but not co-exist [28]. Although the expanding blobs initialization method was not chosen to parallel to natural *Physarum*, the further study of the natural mechanics of *Physarum* may be worthwhile as inspiration for further improvements.

On a more abstract level, similarities can be seen between the visualizations of population size over time, such as in figures 7.5, 7.8 and 7.11, and models of competition, particularely that of Lotka-Volterra [17] ([9] fig. 1 is a clearer visual match). Each plasmodium would represent a species here. In a competitive Lotka-Volterra model in equilibrium each species has both intraspecific competition, limiting its own growth as it becomes larger, and interspecific competition, limiting the population size of the other species. The mechanics at play here are quite similar to those in a stable equilibrium. There is a weak

interplasmodial and a strong intraplasmodial competition, although the intraplasmodial competition occurs through the proxy of city control. This makes for a looser adherence to the idealized equilibrium situation. In addition, the environment introduces a large amount of randomness, as the equations assume an effectively homogeneous environment. The intraspecific competition may be greatly time-lagged, or under certain conditions, such as expansion into an area with no cities, even non-existent. It is possible for the largest plasmodium to control the fewest cities, as in figure 7.12. Nonetheless, a model of the population dynamics at play here may be viable using the Lotka-Volterra model. This would, in turn, allow for a deeper understanding of the tour creation, and so for improvements to the solver.

Chapter 9

Conclusion

A *Physarum* multi-agent model based VRP solver was created. It supports all two-dimensional Euclidean VRPs, which are NP-complete. This multi-agent approach is capable of solving simple VRP's, but not well enough to be competitive with commonly used heuristic solvers. However, it does form a novel method for solving the VRP. Valid tours are formed by an interesting interaction of multiple factors influencing the particle populations. Each plasmodium has impulses to both expand and contract. Due to particles being attracted to the attractant other particles of the same kind deposit, a plasmodium will tend to maintain a whole. Simultaneously, each plasmodium will explore the environment around it, and grow in number where particle density is low. Growth is self-limiting with a dynamic feedback system, such that only plasmodia that are actually in danger of growing too large are throttled.

While performance was not competitive with commonly used solving methods, the techniques used here may be of interest to future research. Particularly the application of a *Physarum* model for splitting a set of vertices into subsets while also solving the TSP is interesting. The solver may be extendable to various other, more complicated, types of VRP. Some of the methods used could also be transferred to the *Physarum* TSP solver. This includes the expanding blobs initialization method, which also forms an interesting parallel to natural *Physarum*. Independent of the initialization method used, the automation of tour reading by analysing the image with a marching squares algorithm, introduced here, could also be applied to the *Physarum* TSP solver and any future solvers of similar problems.

In short, the solver is notable not because of any superior performance but because of its method of operation, combining a direct spatial representation with a stigmergy-based approach. Future research possibilities include extending this solver to more complex problems and applying its techniques to other *Physarum* solvers.

Bibliography

[1] A. Adamatzky. Developing Proximity Graphs By Physarum Polycephalum: Does the Plasmodium Follow the Toussaint Hierarchy? *Parallel Processing Letters*, 19(1): 105–127, 2009. ISSN 1098-6596. doi: 10.1017/CBO9781107415324.004.

[2] A. Adamatzky. Slime mould computes planar shapes. *International Journal of Bio-Inspired Computation*, 4(3):149, 2012. ISSN 1758-0366. doi: 10.1504/IJBIC.2012.047239. URL `http://www.inderscience.com/link.php?id=47239`.

[3] A. Adamatzky, R. Armstrong, B. De Lacy Costello, Y. Deng, J. Jones, R. Mayne, T. Schubert, G. C. Sirakoulis, and X. Zhang. Slime mould analogue models of space exploration and planet colonisation. *JBIS - Journal of the British Interplanetary Society*, 67(7):290–304, 2014. ISSN 0007084X.

[4] M. Aono, Y. Hirata, M. Hara, and K. Aihara. Amoeba-based Chaotic Neurocomputing: Combinatorial Optimization by Coupled Biological Oscillators. *New Generation Computing*, 27(2):129–157, 2009. ISSN 02883635. doi: 10.1007/s00354-008-0058-4.

[5] D. Barbucha. Experimental Study of the Population Parameters Settings in Cooperative Multi-agent System Solving Instances of the VRP. pages 1–28. Springer, Berlin, Heidelberg, 2013.

[6] J. E. Bell and P. R. McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18(1):41–48, jan 2004. ISSN 1474-0346. doi: 10.1016/J.AEI.2004.07.001. URL `https://www.sciencedirect.com/science/article/pii/S1474034604000060`.

[7] G. Clarke and J. W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12(4):568–581, aug 1964. ISSN 0030-364X. doi: 10.1287/opre.12.4.568. URL `http://pubsonline.informs.org/doi/abs/10.1287/opre.12.4.568`.

[8] W. Cook. Concorde TSP Solver, 2016. URL `http://www.math.uwaterloo.ca/tsp/concorde.html`.

[9] K. Z. Coyte, J. Schluter, and K. R. Foster. The ecology of the microbiome: Networks, competition, and stability. *Science (New York, N.Y.)*, 350 (6261):663–6, nov 2015. ISSN 1095-9203. doi: 10.1126/science.aad2602. URL http://www.ncbi.nlm.nih.gov/pubmed/26542567.

[10] G. B. Dantzig and J. H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91, oct 1959. ISSN 0025-1909. doi: 10.1287/mnsc.6.1.80. URL http://pubsonline.informs.org/doi/abs/10.1287/mnsc.6.1.80.

[11] B. Eksioglu, A. V. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4): 1472–1483, 2009. ISSN 03608352. doi: 10.1016/j.cie.2009.05.009. URL http://www.sciencedirect.com/science/article/pii/S0360835209001405.

[12] V. Evangelidis, J. Jones, N. Dourvas, M.-A. Tsompanas, G. C. Sirakoulis, and A. Adamatzky. Physarum machines imitating a Roman road network: the 3D approach. *Scientific Reports*, 7(1):7010, dec 2017. ISSN 2045-2322. doi: 10.1038/s41598-017-06961-y. URL http://www.nature.com/articles/s41598-017-06961-y.

[13] C. L. Fleming, S. E. Griffis, and J. E. Bell. The effects of triangle inequality on the vehicle routing problem. *European Journal of Operational Research*, 224(1):1–7, 2013. ISSN 03772217. doi: 10.1016/j.ejor.2012.07.005. URL http://www.sciencedirect.com/science/article/pii/S0377221712005267.

[14] M. Gendreau, A. Hertz, G. Laporte, Gendreau, A. Hertz, G. Laporte, and M. Gendreau. A Tabu Search Heuristic for the Vehicle Routing Problem. *Management Science*, 40(10):1276–1290, oct 1994. ISSN 0025-1909. doi: 10.1287/mnsc.40.10.1276. URL http://www.jstor.org/stable/10.2307/2661622%5Cn.

[15] Golden, Bruce L., Arjang A. Assad and E. A. Wasil. Routing vehicles in the real world: applications in the solid waste, beverage, food, dairy, and newspaper industries. The vehicle routing problem. Society for Industrial and Applied Mathematics APA. *InformationSciences*, 176(18):2698–2712, 2001. ISSN 00200255. doi: 10.1016/j.ins.2005.11.012.

[16] Google. Google Optimization Tools: Vehicle Routing Problem, 2017. URL https://developers.google.com/optimization/routing/tsp/vehicle_routing.

[17] N. J. Gotelli. Competition. In *A primer of ecology*, chapter 5, pages 100–124. Sinauer Associates Incorporated, 1995. URL https://www.cabdirect.org/cabdirect/abstract/19970502464.

[18] J. Jones. The emergence and dynamical evolution of complex transport networks from simple low-level behaviours. *International Journal of Unconventional Computing*, 6 (2):125–144, 2010. ISSN 15487199. URL http://arxiv.org/abs/1503.06579.

[19] J. Jones. Characteristics of pattern formation and evolution in approximations of Physarum transport networks. *Artificial Life*, 16(2):127–153, apr 2010. ISSN 1064-5462. doi: 10.1162/artl.2010.16.2.16202. URL http://www.mitpressjournals.org/doi/10.1162/artl.2010.16.2.16202.

[20] J. Jones. Influences on the formation and evolution of Physarum polycephalum inspired emergent transport networks. *Natural Computing*, 10(4):1345–1369, dec 2011. ISSN 15677818. doi: 10.1007/s11047-010-9223-z. URL http://link.springer.com/10.1007/s11047-010-9223-z.

[21] J. Jones. Applications of multi-agent slime mould computing. *International Journal of Parallel, Emergent and Distributed Systems*, 31(5):420–449, 2016. ISSN 1744-5760. doi: 10.1080/17445760.2015.1085535. URL http://www.tandfonline.com/doi/full/10.1080/17445760.2015.1085535.

[22] J. Jones and A. Adamatzky. Computation of the travelling salesman problem by a shrinking blob. *Natural Computing*, 13(1):1–16, 2014. ISSN 15677818. doi: 10.1007/s11047-013-9401-x.

[23] M. Khachay and R. Dubinin. Approximability of the d-dimensional euclidean capacitated vehicle routing problem. *AIP Conference Proceedings*, 1776:1–5, 2016. ISSN 15517616. doi: 10.1063/1.4965323.

[24] S. N. Kumar. A Survey on the Vehicle Routing Problem and Its Variants. *Intelligent Information Management*, 04(03):66–74, 2012. ISSN 2160-5912. doi: 10.4236/iim.2012.43010. URL http://dx.doi.org/10.4236/iim.2012.43010 http://www.scirp.org/journal/iim.

[25] G. Laporte and Y. Nobert. A Branch and Bound Algorithm for the Capacitated Vehicle Routing Problem. *OR Spektrum*, 5:77–85, 1983. ISSN 0171-6468. doi: 10.1007/BF01720015.

[26] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm, 1987. ISSN 00978930. URL http://portal.acm.org/citation.cfm?id=37422.

[27] L. Masi and M. Vasile. Optimal multi-objective discrete decision making using a multidirectional modified Physarum Solver. In *EVOLVE 2012 International Conference.*, 2012. URL http://strathprints.strath.ac.uk/41573/.

[28] L. S. Olive. Myxogastria (Myxomycetes). In *The Mycetozoans*, chapter 4, pages 99–156. Academic Press, New York, New York, USA, 1975. ISBN 9780125262507. doi: 10.1016/B978-0-12-526250-7.50008-9. URL http://linkinghub.elsevier.com/retrieve/pii/B9780125262507500089.

[29] C. H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, 1977. ISSN 03043975. doi: 10.1016/0304-3975(77)90012-3. URL http://www.sciencedirect.com/science/article/pii/0304397577900123.

[30] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved Branch-Cut-and-Price for capacitated vehicle routing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8494 LNCS:393–403, 2014. ISSN 16113349. doi: 10.1007/978-3-319-07557-0-33.

[31] J. Y. Potvin. A review of bio-inspired algorithms for vehicle routing. *Studies in Computational Intelligence*, 161:1–34, 2009. ISSN 1860949X.

[32] L. Snyder. VRP Solver, 2004. URL coral.ise.lehigh.edu/larry/software/vrp-solver/.

[33] A. Tero, S. Takagi, T. Saigusa, K. Ito, D. P. Bebber, M. D. Fricker, K. Yumiki, R. Kobayashi, and T. Nakagaki. Rules for Biologically Inspired Adaptive Network Design. *Science*, 327(5964):439–442, jan 2010. ISSN 0036-8075. doi: 10.1126/science.1177894. URL http://www.ncbi.nlm.nih.gov/pubmed/20093467 http://www.sciencemag.org/cgi/doi/10.1126/science.1177894.

[34] P. Toth and D. Vigo. *Vehicle Routing Problem, Methods, and Application*. 2014. ISBN 9781611973587.

[35] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu. scikit-image: image processing in Python. *PeerJ*, 2:e453, jun 2014. ISSN 2167-8359. doi: 10.7717/peerj.453. URL https://peerj.com/articles/453.

[36] T. Wąsiński. Vehicle routing problem in Py, 2015. URL https://github.com/wasinski/VRP.

Part IV

Appendix

Chapter 10

Results per problem

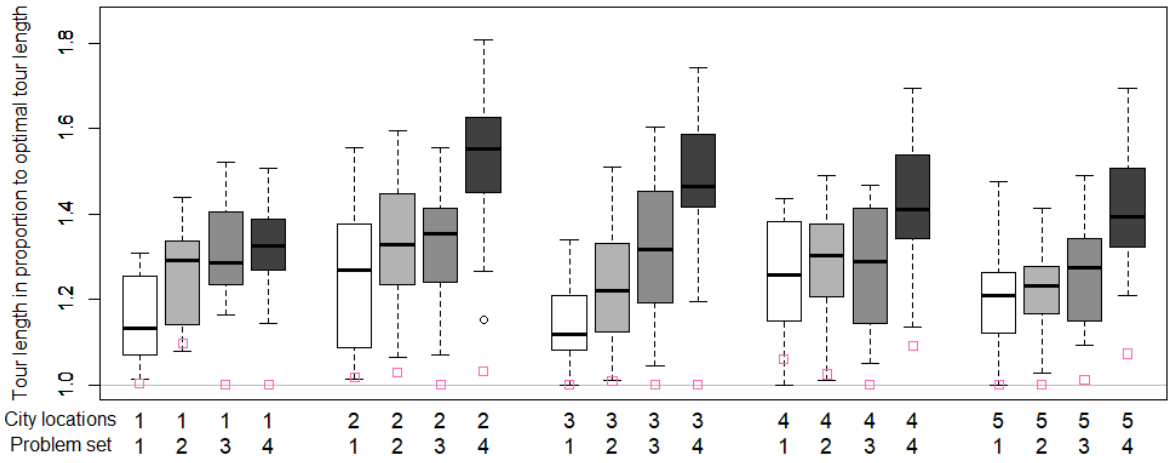


Figure 10.1: *Physarum* VRP solver tour lengths per problem for city location sets 1 to 5, for problem sets 1 to 4. 20 runs per condition. Pink squares indicate the OR-Tools solution. All are normalized to the be in proportion to the optimal solution.

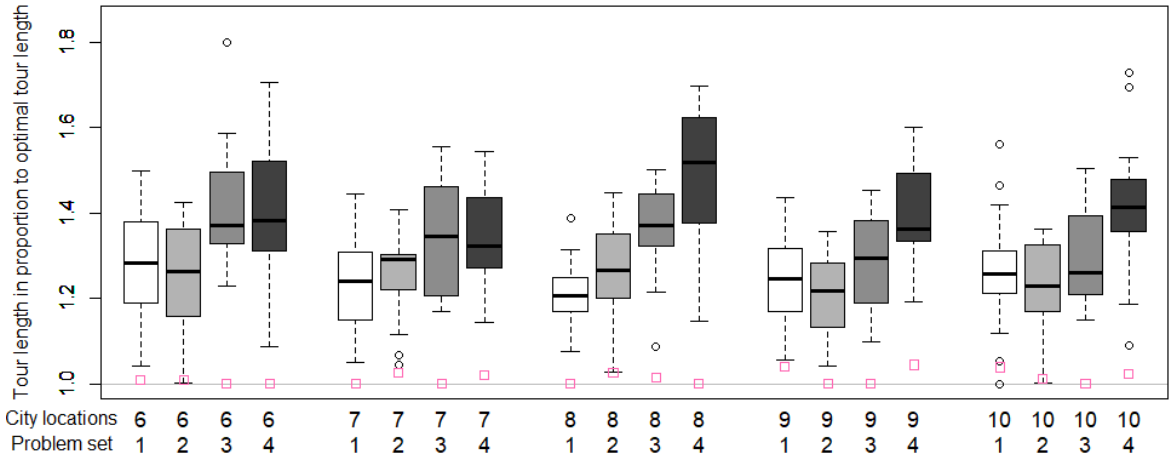


Figure 10.2: *Physarum* VRP solver tour lengths per problem for city location sets 6 to 10, for problem sets 1 to 4. Pink squares indicate the OR-Tools solution. All are normalized to the be in proportion to the optimal solution.

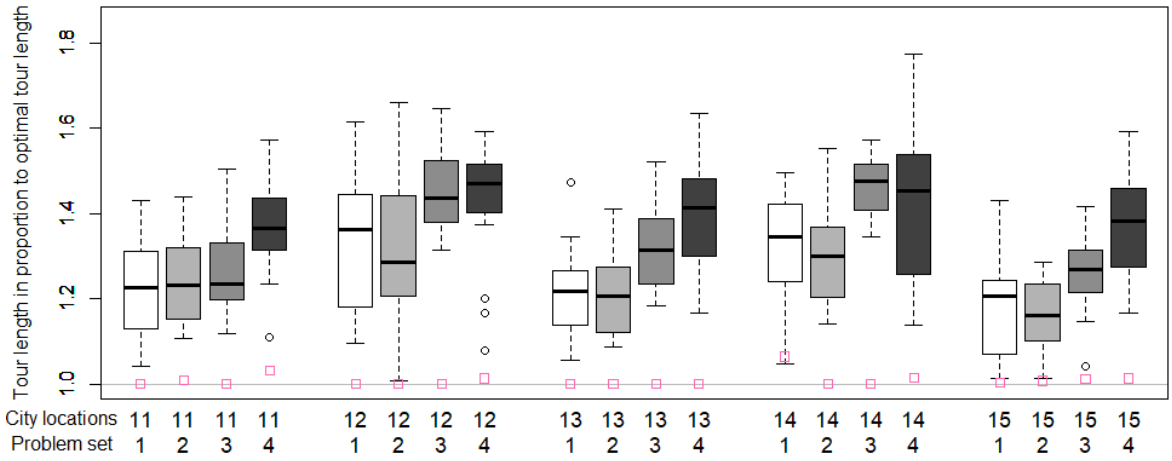


Figure 10.3: *Physarum* VRP solver tour lengths per problem for city location sets 11 to 15, for problem sets 1 to 4. 20 runs per condition. Pink squares indicate the OR-Tools solution. All are normalized to the be in proportion to the optimal solution.

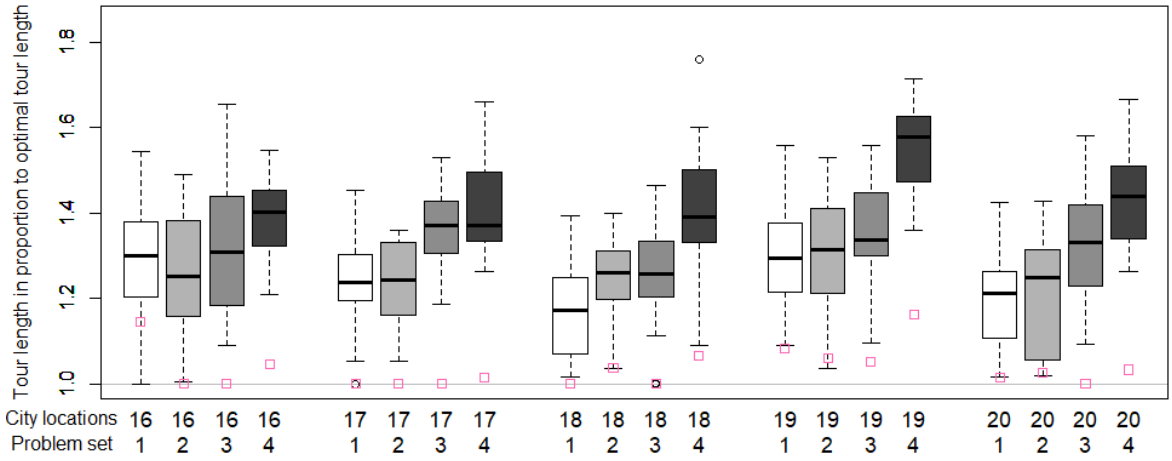


Figure 10.4: *Physarum* VRP solver tour lengths per problem for city location sets 16 to 20, for problem sets 1 to 4. 20 runs per condition. Pink squares indicate the OR-Tools solution. All are normalized to the be in proportion to the optimal solution.

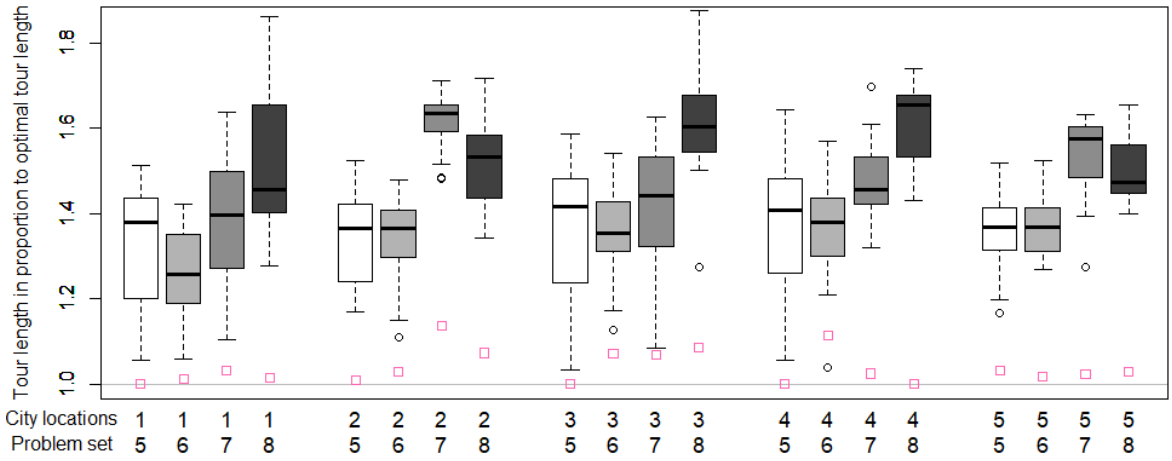


Figure 10.5: *Physarum* VRP solver tour lengths per problem for city location sets 1 to 5, for problem sets 5 to 8. 20 runs per condition. Pink squares indicate the OR-Tools solution. All are normalized to the be in proportion to the solutions of Snyder's solver.

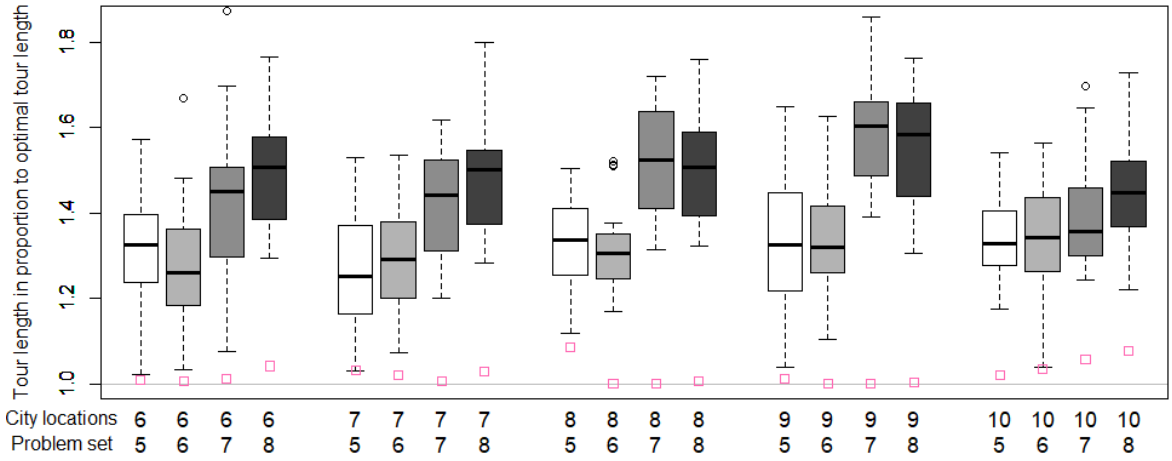


Figure 10.6: *Physarum* VRP solver tour lengths per problem for city location sets 6 to 10, for problem sets 5 to 8. 20 runs per condition. Pink squares indicate the OR-Tools solution. All are normalized to the be in proportion to the solutions of Snyder's solver.

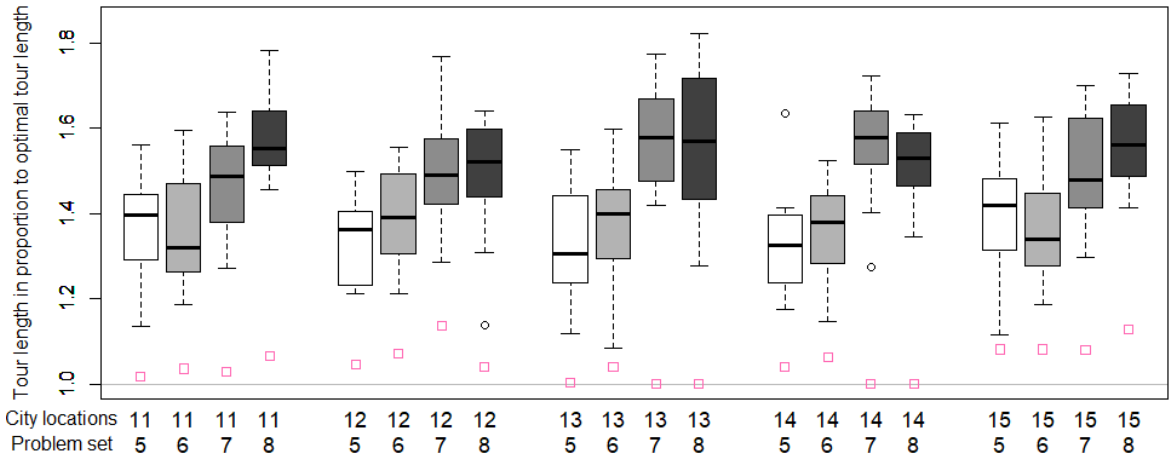


Figure 10.7: *Physarum* VRP solver tour lengths per problem for city location sets 11 to 15, for problem sets 5 to 8. 20 runs per condition. Pink squares indicate the OR-Tools solution. All are normalized to the be in proportion to the solutions of Snyder's solver.

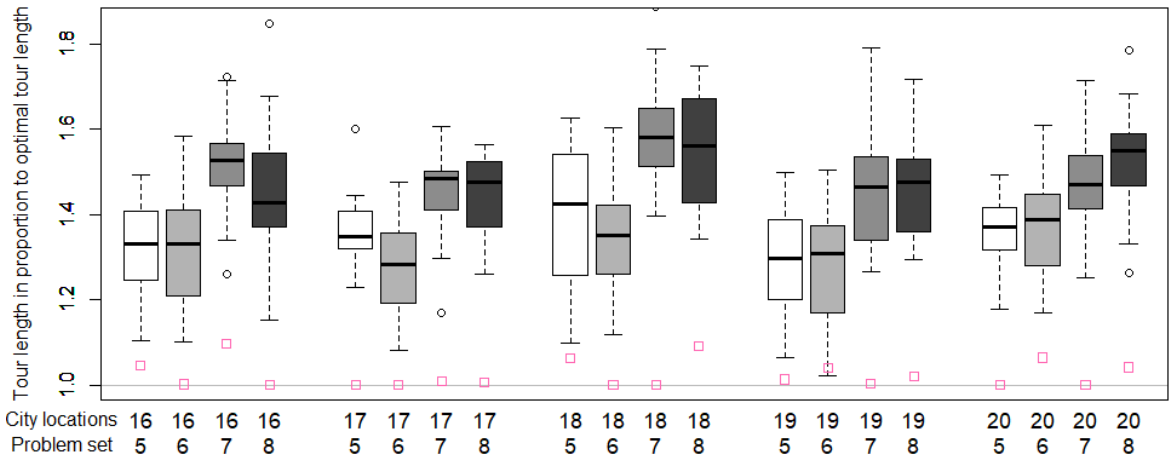


Figure 10.8: *Physarum* VRP solver tour lengths per problem for city location sets 16 to 20, for problem sets 5 to 8. 20 runs per condition. Pink squares indicate the OR-Tools solution. All are normalized to the be in proportion to the solutions of Snyder's solver.

Chapter 11

Random solution generation

This chapter describes the generation of the random solutions. These are generated separately for each individual problem, using the problem parameters and city locations of that problem. The solutions are generated by giving a number of cities, randomly selected from those not yet assigned, to each vehicle. The number is random, but at least the minimum number of cities that must be assigned to each vehicle in a valid solution, e.g. if there are 2 vehicles of capacity 6 in a problem with 10 cities, a vehicle can visit no fewer than 4 cities. At most this number is equal to the capacity. This process occurs sequentially for each vehicle in the problem. If, at any point, the number of cities left to assign is lower than the capacity, the next vehicle will be assigned all remaining cities. This process does not guarantee valid solutions. For example, if three vehicles are used, it is possible for the first two to be assigned so few cities that the third vehicle can no longer be assigned all remaining cities. Invalid solutions are rejected, and new solutions generated until the desired number of valid solutions has been created.

Once the solution has been deemed valid, the total tour length is calculated as normal, simply adding the lengths of all parts of each tour and then summing each tour. Because all cities are chosen randomly from the set of remaining cities, each tour visits its cities in a random order. All tours still start and end at the depot, as typical.

Chapter 12

Snyder's solver

Table 12.1: Parameters and chosen values in Snyder's VRP solver

Parameter	Value
Randomization depth	30
Randomization iterations	30
Improvement heuristics	All used
OR-Opt Options	All on

Snyder's solver was developed in 2003 at Lehigh University, USA by Lawrence V. Snyder [32]. The latest version, 1.3, was used. The program settings used are described in the table. In short, Snyder's solver uses a randomized adaptation of the Clarke-Wright savings algorithm, expanded with several improvement heuristics [32]. The choice of routes to merge is semi-randomized for several steps. This is then repeated from the start several times, with the best outcome being used. Several improvement heuristics are also used, such as random swaps between routes. A full description of the functioning of the solver is included in its accompanying readme file.