Agent Based Models of Competition and Collaboration.

Harry James Goldingay

Doctor Of Philosophy

– Aston University –

March 2010

Aston University

Agent Based Models of Competition and Collaboration.

Harry James Goldingay

Doctor Of Philosophy, 2010

Thesis Summary

Swarm intelligence is a popular paradigm for algorithm design. Frequently drawing inspiration from natural systems, it assigns simple rules to a set of agents with the aim that, through local interactions, they collectively solve some global problem. Current variants of a popular swarm based optimization algorithm, particle swarm optimization (PSO), are investigated with a focus on premature convergence. A novel variant, dispersive PSO, is proposed to address this problem and is shown to lead to increased robustness and performance compared to current PSO algorithms.

A nature inspired decentralised multi-agent algorithm is proposed to solve a constrained problem of distributed task allocation. Agents must collect and process the mail batches, without global knowledge of their environment or communication between agents. New rules for specialisation are proposed and are shown to exhibit improved efficiency and flexibility compared to existing ones. These new rules are compared with a market based approach to agent control. The efficiency (average number of tasks performed), the flexibility (ability to react to changes in the environment), and the sensitivity to load (ability to cope with differing demands) are investigated in both static and dynamic environments. A hybrid algorithm combining both approaches, is shown to exhibit improved efficiency and robustness.

Evolutionary algorithms are employed, both to optimize parameters and to allow the various rules to evolve and compete. We also observe extinction and speciation. In order to interpret algorithm performance we analyse the causes of efficiency loss, derive theoretical upper bounds for the efficiency, as well as a complete theoretical description of a non-trivial case, and compare these with the experimental results. Motivated by this work we introduce agent "memory" (the possibility for agents to develop preferences for certain cities) and show that not only does it lead to emergent cooperation between agents, but also to a significant increase in efficiency.

Keywords: Evolutionary Computation, Optimization, Swarm Intelligence, Task Allocation

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost I would like to thank my supervisor, Dr. Jort van Mourik, who provided much needed advice and insight into the work herein, and without who this thesis would not have been written.

I am grateful to all the staff at Aston University and, in particular, to Vicky Bond for ensuring that the real world didn't intrude too much upon my research, and to Alex Brulo whose technical support enabled the experimental side of my work.

I would also like to express gratitude to Dr. Peter Tiňo, Dr. A. Ekart and Dr. Juan Neirotti for stimulating discussions, useful suggestions, and careful reading of various elements of this manuscript.

This thesis was made possible due to funding from the EPSRC and due to support and and equipment from the Non-linearity and Complexity Research Group at Aston University. I also extend my thanks to my friends and colleagues within the research group, and within the university, who have made the time spent working here such a pleasant experience.

Finally, I am deeply grateful to my family, and particularly to my parents, for providing encouragement and support throughout my education.

1 Introduction

CONTENTS

Swarm intelligence is a popular paradigm for algorithm design. Drawing inspiration from natural systems, it assigns simple rules to a set of agents with the aim that, thorough local interactions, they collectively solve some global problem.

In this thesis, we consider a nature inspired, decentralised, multi-agent algorithm for distributed task allocation. We propose a set of new rules for task specialisation and test them on an idealised model of large scale task allocation, the mail retrieval problem. We test examine their performance both qualitatively and, through the use of an evolutionary algorithm in which the various rules can evolve and compete, quantitatively. In order to interpret algorithm performance we analyse the causes of efficiency loss, and derive a complete theoretical model of the system under non-trivial conditions.

The best new rule found in this manner is compared with a market based approach to agent control. We focus on efficiency, flexibility, and robustness under a range of conditions. We also combine the threshold and market based approaches into a hybrid algorithm, which exhibits improved efficiency and robustness. Again, we use an evolutionary algorithm to quantitatively compare all algorithms in a range of representative environments. We also derive theoretical upper bounds for the efficiency, and compare these with the experimental results.

Motivated by this work, we propose a threshold based algorithm in order to maximise the overall efficiency. We show that memory, i.e. the possibility for agents to develop preferences for certain cities, not only leads to emergent cooperation between agents, but also to a significant increase in efficiency (above the theoretical upper limit for any memoryless algorithm).

Finally, we investigate current variants of a popular swarm based optimization algorithm, particle swarm optimisation (PSO), with with a focus on premature convergence. We propose a novel variant, dispersive PSO, to address this problem and is shown to lead to increased robustness and performance compared with current PSO algorithms.

1.1 Motivation

Firstly, nature-inspired rules are partially constrained by the circumstances, resources and requirements under which they evolved. When dealing with artificial

systems, these conditions may be substantially different. As such, when looking at how a nature-inspired algorithm can be improved, it is important to analyse what scope for improvement there is within the current framework. In this way, limitations within the current framework can be systematically addressed and appropriate modifications to the framework designed, in order to maximise performance.

Emergent behaviour in swarm intelligence systems is the foundation of their success. However, due to the high degree of difficulty in specifying individual rules which lead to exactly the desired behaviour, good rules frequently lead to elements of undesirable behaviour. An example of such behaviour is "premature convergence" in PSO, in which the entire swarm converges to a single point, eliminating all future search capacity. The modification of standard behaviour required to eliminate this would remove the capacity for local searches, compromising overall performance. As such a conditional modification, which allows the swarm to search normally until premature convergence is threatened and then changes agent behaviour to avoid it, is desirable.

1.2 Objectives

The main objectives of this thesis are:

- To investigate and improve the current nature inspired rule sets for threshold based task allocation.
- To compare threshold based task allocation algorithms with leading current methods under a range of circumstances in order to identify which methods are best suited to particular conditions.
- To theoretically model the problem of distributed task allocation upon which our work is based, and to analyse these models in order to gain insight into an "ideal" decentralised distributed task allocation algorithm.
- To analyse the causes of premature convergence in the PSO algorithm, and to design an algorithm to systematically reverse them in order to regain search performance.

1.3 Methodology

As the self organising behaviour exhibited by social insects appears in (large) colonies, it seems natural to consider the performance of our task allocation model with a large population of agents. Not only does the large system size have the advantage of removing finite size effects (such as large fluctuations both inside and in between different runs), but Anderson and Ratnieks [1, 2] have also shown that a specific form of collective behaviour involving direct cooperation between agents is only efficient in large systems. Furthermore, Dornhaus et al. [18] hypothesise that the simulation of honeybee behaviour that they have investigated, did not produce realistic behaviour because of the small size of the system used. Hence, in this chapter we consider the behaviour of the system mostly from a population dynamics perspective where the average behaviour is of greater importance than the individual performance of an agent. Nevertheless, we also investigate the influence of (small) system size on the overall efficiency and fluctuations thereof.

When testing our PSO variant, we compare its performance with a good current PSO variant, as well as with its nearest competitor. In order to do this we attempt to identify a set of standard test functions, from the literature on evolutionary algorithms, which are representative of a wide range of problem classes. As PSO performance can be fairly chaotic we compare the algorithms' performances with respect to performance and robustness as an average of a large number of runs.

Note that all simulations are implemented in C++ and are performed on a linux-PC cluster.

1.4 Contributions

The main contributions of this thesis are:

- The development of a constrained task allocation problem which allows analysis of large scale, decentralised, task allocation algorithms under a range of conditions.
- The development and analysis of new specialisation rules in threshold based task allocation.

- A comparison of these new threshold based approaches with a current market based approach. We analyse behaviour in terms of both absolute performance (following optimisation) and the conditions under which such performance can be obtained.

- The introduction of a stigmergic "memory" based mechanism for task allocation, allowing close to perfect performance in our task allocation problem due to emergent cooperation.

- An analysis of flaws in a standard PSO variant for avoiding premature convergence and the introduction of an algorithm, dispersive PSO, which allows for greater robustness when solving this problem.

1.5 Structure

The structure of the thesis is as follows. In chapter 2 we give some background to our work, discussing swarm intelligence and introducing the evolutionary algorithms which we use in our work. In chapter 3 we introduce a model of distributed task allocation and investigate methods for solving it in the general case, whereas chapter 4 looks at a specific case of the problem and how, under certain conditions, we can achieve an close to perfect solution. In chapter 5 we discuss methods of dealing with premature convergence in PSO, discussing a current method and introducing our own variant, before performing a comparative study. Chapter 6 summarises our findings and gives an outlook to future work.

2 Background

CONTENTS

2.1 Swarm Intelligence

The solution of problems using multiple software systems, distributed artificial intelligence, is an important and developing field of research. It has been applied to problems from prioritising information given to fighter pilots to assisting with the diagnosis of problems in the CERN particle accelerator [12]. In [19] it is noted that distribution of A.I. leads to the following useful properties.

- Parallel problem solving can lead to more efficient use of resources.
- Problems which are spatially distributed can cut down on their need for communication by being able to make independent decisions
- The modularisation of software makes it easier to manage.
- Systems which are composed of multiple intelligent entities can be simulated well by combining simulations of these individuals.

Aside from cost factors their may be inherent barriers in communication which render centralised control ether impractical or impossible.

We shall be concerned with the subset of distributed A.I. dealing with autonomous agents. Autonomous agents act within an environment using the properties that

- They have some ability to collect data from their environment.
- They use this data to decide upon some action based on simple rules with no outside input.
- The action which they take can affect their environment.

The aim is that they are designed such that, in concert, their actions combine to complete some task. Also, as they have no centralised means of correcting their behaviour, the system as a whole must be robust and capable of adapting to changes in its environment. In such a situation self organisation, the ability of the system to adapt itself to its problem or environment without centralised control, is clearly a desirable attribute.

A system which closely shares the properties of autonomous agents and already possesses self organisation is that of social insects. For instance, in [34], Huang et al. show that in honeybee colonies which have too few adult members to forage effectively, some juveniles develop faster to fill the adult roles. The number of juveniles undergoing this hastened development was shown to decrease with the increase of the proportion of foragers. A mechanism suggested to explain this and other similar social insect behaviours is stigmergy. Introduced by Grassé in 1959, the hypothesis is that members of the colony perform actions based on their environment, thus altering the environment. This altered environment would then trigger another set of, possibly different, environment altering actions. This repeating chain of action and reaction has evolved in such a way that it gives rise to apparently cooperative behaviour which leads to the accomplishment of useful tasks. The similarity of autonomous agents and social insects has been exploited, and algorithms based on this similarity fall under the heading of **swarm intelligence**. Simulations with stigmergic rules have been run and behaviour comparable to that seen in nature has been observed [10],[18].

Of chief importance to us is the fact that any simulation which accurately reproduces the behaviours described must have the property of self organisation. This means that if we wish to solve a distributed problem and can find a suitable analogue for it in nature which has been solved by social insects, modelling the behaviour of these insects can lead to novel and effective solutions. Both [16] and [63] describe the use of mobile agents with rules based on path finding methods used by ants while foraging to create network routing algorithms. Both algorithms are not only highly adaptable to changes in network traffic, but also outperform a variety of other algorithms in realistic conditions. Other applications, examined in detail by Bonabeau et al. in [9], include

- Good solutions to shortest path problems such as the travelling salesman problem can be arrived at using the ACO metaheuristic, which is based on ant foraging behaviour.
- Systems inspired by brood sorting behaviour have provided good approaches to graph partitioning problems.

- An algorithm inspired by nest building behaviour in termites has been used to improve the performance of self assembling robots.

The complexity of natural systems combined with the number of possible applications ensure that there are many open questions left within the field.

2.2 Evolutionary Optimization

Another field which has drawn inspiration from natural systems is evolutionary optimization, the use of **evolutionary algorithms (EAs)** to optimise a given system. EAs are heuristic methods which use populations of candidate solutions, coupled with some nature-inspired rule set, to perform a process of iterative improvement. These algorithms show good robustness and performance on a wide range of problems [23].

In order to engage in optimization, we need to define three things:

- A **search-space**, $\mathcal{S}$, corresponding to a set of inputs which we can change.
- The set of constraints on these inputs.
- Some way of measuring the quality of a particular configuration of variables.

Given a specific input, $\vec{x} \in \mathcal{S}$, we denote its quality using a **cost function** $f(\vec{x})$. In general this function can be such that $f : \mathcal{S} \to \mathbb{R}^n$. However, in this thesis we consider only **single-objective optimization** in which $f : \mathcal{S} \to \mathbb{R}$. It is important to note that f is not required to be a mathematical function (it could be the output of some industrial process or even human input) it is just representative of solution quality.

For the duration of this thesis we assume that $\mathcal{S}$ has a simple (rectangular) set of bounds, so that a component in the i^{th} dimension of the search space must lie in the interval $[b_i^{min}, b_i^{max}]$. While this is an oversimplification for real-world problems, it allows us to compare our algorithms with in a similar setting to the literature (see, for instance [69; 71]). In addition for the main optimisation algorithm studied in this thesis, particle swarm optimisation, optimal behaviour at a boundary is an open problem even for rectangular boundaries [33]. The study of this problem is beyond the scope of this thesis.

Table 2.1: Use of paradigms in evolutionary algorithms

EA	Search Space	Search Mechanism	Improvement Mechanism
Genetic Algorithm	Chromosomes	Mutation and Crossover	Natural Selection
Ant Colony Optimization	Foraging paths	Divergence from pheromones	Pheromone decay
Particle Swarm Optimization	Positions in swarm	Attraction to neighbours	Memory

The general purpose of any single-objective optimisation algorithm is to take a set of inputs and to either minimise or maximise some single output based on these inputs. It is convenient to think of this in terms of minimising our cost function as this can be done without loss of generality because minimising f is equivalent to maximising $-f$. More formally, we aim to find some $\vec{x} \in \mathcal{S}$ such that $f(\vec{x}) \leq \theta$

- **Ideally** for all thresholds $\theta \in \mathcal{R}$.
- **In practice** for some acceptable threshold $\theta \in \mathcal{R}$.

where $\mathcal{R}$ is the range of f.

Typically when minimising a cost function, an EA is not concerned with the exact detail of the natural behaviour which inspired it, but instead in using the paradigm behind the behaviour to design an efficient algorithm. In table 2.1 we give some examples of these paradigms, broken down in to three categories. The representation of a candidate solution in the search space is some object or phenomenon which appears in nature, and which undergoes improvement through a defined process. This process is then encoded in the search and improvement mechanisms, with the search process moving the object in the search space. The improvement mechanism aims to in some way bias the system into producing, or preserving, those moves which improve the overall state of the system.

We will now discuss two evolutionary algorithms used within this thesis: evolution strategies and particle swarm optimization.

2.2.1 Evolution Strategies

Evolution strategies (ES) is an algorithm [4] based on a similar paradigm to **genetic algorithms (GAs)**, biological evolution. To optimise a cost function f ES uses a population of real-valued individuals with variables $(\mathbf{x}, \boldsymbol{\sigma}, F)$. Here the elements of $\mathbf{x}$, referred to as the *object parameters*, are the inputs into the function

we wish to optimise. $\boldsymbol{\sigma}$ are known as *strategy parameters* and control the mutation of $\mathbf{x}$. Mutation is also applied to the strategy parameters, allowing them to self-adapt to the structure of the fitness space. The **fitness**, $F_m = f(\mathbf{x}_m)$, is a measure of the quality of $\mathbf{x}$ for optimising our function f.

An ES algorithm starts generation g with a set of μ **parents**, $\mathcal{P}(g)$ and proceeds to create a set of ℓ offspring, $\mathcal{O}(g)$. Each offspring agent is selected by choosing ρ (called the mixing number) parents at random from $\mathcal{P}(g)$ and *recombining* them into a single individual. This new individual is then **mutated** according their strategy parameters. These populations undergo **selection**, keeping the fittest μ individuals as a new generation of parents.

In this thesis we will not discuss on recombination schemes as we always take $\rho = 1$ and, as such, when a parent $(\mathbf{x}, \boldsymbol{\sigma}, F)$ is chosen to produce offspring $(\tilde{\mathbf{x}}, \tilde{\boldsymbol{\sigma}}, \tilde{F})$, we copy the parent's variables. For standard ES this initial offspring is then mutated, component-wise for each dimension i, and its fitness is evaluated according to the following procedure:

1. $\tilde{\sigma}_i = \sigma_i \cdot e^{\xi_i}$
2. $\tilde{x}_i = x_i + \tilde{\sigma}_i \cdot z_i$
3. $\tilde{F} = f(\tilde{\mathbf{x}})$

where $\xi_i \sim N(0, \frac{1}{\sqrt{d}})$ and $z_i \sim N(0, 1)$ are independent random numbers and d is the number of dimensions of our object parameter $\tilde{\boldsymbol{x}}$.

Once a complete set of ℓ offspring has been created, the population undergoes selection during which those μ individuals with the highest fitness are deterministically selected from either

- the current set of offspring ($\mathcal{O}(g)$). This is known as the **$(\boldsymbol{\mu}, \boldsymbol{\ell})$-ES**.
- the current set of offspring and parents ($\mathcal{P}(g) \cup \mathcal{O}(g)$). This is known as the **$(\boldsymbol{\mu} + \boldsymbol{\ell})$-ES**.

Once selected, they are used as the new generation of parent agents $\mathcal{P}(g+1)$. The repeated iteration of this process comprises an ES algorithm.

Covariance matrix adaption evolution strategies (CMA-ES) is a variant of ES introduced by Hansen and Ostermeier [29]. Rather than considering strategy

parameters for each individual and for each dimension, CMA works with a single distribution defined by a weighted population centre and a full covariance matrix. All individuals offspring are created by sampling this distribution. The covariance matrix is updated using information gathered by the whole population in such a way that it encourages mutation from the mean in the gradient direction. This improves its performance as a local optimizer, but not its global search performance [29]. As our application of the ES algorithm in this thesis does not require fast local search, we prefer standard ES over CMA-ES for simplicity.

2.2.2 Particle Swarm Optimization

Particle swarm optimisation (PSO) is a swarm intelligence based algorithm, [38], which has emerged as a viable continuous optimisation algorithm. Similarly to the GA it is comprised of a population of individuals (or particles) moving in, and evaluating, the fitness landscape. However, rather than relying on random moves to improve their fitness, these particles retain information about good locations and share them with the rest of the swarm. Particles are attracted to these good locations allowing efficient searching of the fitness space. This often allows PSO to find comparable solutions to naive GAs in fewer function evaluations [30]. The efficiency of PSO, combined with it retaining the relative simplicity of a GA, makes it a good general purpose search heuristic. It has been applied to the design and optimisation of a wide range of systems, including:

- antenna design,
- control systems,
- electronics,
- video analysis,

For a comprehensive overview of the applications of PSO, see [52].

PSO uses a population of n particles, whose d-dimensional positions $\vec{x}$ are used as candidate solutions to minimise the objective function and who move according to certain rules. These rules were inspired by the swarming/flocking behaviour of various animals. These animals are hypothesised to gain a competitive advantage in

the search for resources (e.g. food, roosts) by socially sharing information [38]. In a canonical PSO algorithm individuals remember their best previously discovered location and share it with the rest of the swarm. The population is then attracted towards these good locations, replacing points of attraction with better ones as they are discovered.

2.2.2.1 Details

In the standard literature a swarm is defined at time t by the following set of variables:

$$\{(\vec{x}_i(t), \vec{v}_i(t), \vec{h}_i(t)), \quad i = 1, .., n\} \tag{2.1}$$

where, for a given particle i, we have

- $\vec{x}_i(t)$, the particle's **position**. This is the point selected by the particle as a candidate solution for minimising the cost function. $f(\vec{x}_i(t))$ is evaluated at every time-step.

- $\vec{v}_i(t)$, the particle's **velocity**. This is a standard discrete-time position update, with the particle' next position defined as

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t) \tag{2.2}$$

  Because positions are used as candidate solutions, how velocity is calculated in effect determines the algorithm's search behaviour.

- $\vec{h}_i(t)$, the particle's **historic best** position. Each particle remembers the best (i.e. lowest cost) position it has visited over the course of a run. Formally, this can be written as

$$\vec{h}_i(t) = \underset{\vec{x}_i(\tau)}{\operatorname{argmin}} f(\vec{x}_i(\tau)), \quad \tau = 1, ..., t \tag{2.3}$$

  Just as a particle's velocity determines changes in its position, the memory of these good positions is used to drive changes in a particle's velocity.

Additionally, in order to extend the generality of our definitions we introduce a new variable:

- $\vec{a}_i(t)$, the particle's **attractor**. Similarly to its historic best position an attractor is a low cost position discovered by a particle over the course of a run and used to drive velocity changes. However, unlike with a historic best position, it is possible (depending on the details of the algorithm) for particles to forget their attractors and set them to positions of higher cost.

Given a position $\vec{x}_i(t+1)$ and previous attractor $\vec{a}_i(t)$ the update equation for a particle's attractor is given by

$$\vec{a}_i(t+1) = \begin{cases} \vec{x}_i(t+1) & \textbf{if } f(\vec{x}_i(t+1)) < f(\vec{a}_i(t)) \textbf{ and } \vec{x}_i(t+1) \text{ is } \textit{valid}, \\ \vec{a}_i(t) & \textbf{otherwise} \end{cases} \tag{2.4}$$

Unless specified by a non-standard algorithm, we consider all positions $\vec{x}_i(t)$ which meet the the constraints on the search space to be *valid*. As such, for canonical PSO algorithms and their derivatives, a particle's attractor $\vec{a}_i(t)$ is *identical to its historic best position*, $\vec{h}_i(t)$.

This gives us our full variable set

$$\{(\vec{x}_i(t), \vec{v}_i(t), \vec{h}_i(t), \vec{a}_i(t)), \quad i = 1, .., n\} \tag{2.5}$$

As we shall discuss, these variables determine the areas in which PSO searches. As such, it can be considered that the overall search behaviour of PSO is driven by changes in these variables.

2.2.2.2 Movement

Given a fixed set of variables, the ultimate aim of PSO is to discover a lower cost position than the current **global historic best** position. This is the best position evaluated by the swarm so far and is formally defined as

$$\hat{h}(t) = \underset{\vec{h}_i(t)}{\operatorname{argmin}} f(\vec{h}_i(t)), \quad i = 1, ..., n \tag{2.6}$$

In order to do this, PSO attempts to assign velocities to its particles such that they move towards promising locations in the search space. In standard PSO, particles use two positions as clues to these promising locations: their historic best position (represented here by the particle's attractor), and a socially shared position taken from the historic best positions of other particles. The original movement

update equation is given by

$$\vec{v}_i(t) = \underbrace{\vec{v}_i(t-1)}_{\text{Inertial}} + \underbrace{c_1\ \vec{r}_1\ \otimes\ (\vec{a}_i(t) - \vec{x}_i(t))}_{\text{Cognitive}} + \underbrace{c_2\ \vec{r}_2\ \otimes\ (\hat{h}(t) - \vec{x}_i(t))}_{\text{Social}} \tag{2.7}$$

where $\otimes$ is the component wise vector product (i.e. $\vec{a} \otimes \vec{b} = (a_1 b_1, a_2 b_2, ...)$, note that this is not an inner product as it must produce a new velocity *vector*). $\vec{r}_1, \vec{r}_2$ are d dimensional vectors with components drawn independently from $U[0,1]$, and c_1 and c_2 are constants determining the relative magnitude of the "cognitive" and "social" parts of the equation respectively. This velocity is then *clamped* to decrease the chance of a particle leaving the search space. In practice, this typically means that no dimension of $\vec{v}$ is permitted to exceed the size of the corresponding search space dimension, with those that do being artificially limited.

Equation (2.7) can be considered to have two purposes. Expansion, governed by the velocity term and local search around discovered positions governed by the cognitive and social terms. In addition, the relative values of c_1 and c_2 control convergence behaviour. Prioritising the cognitive term will cause the swarm to spend more of its time exploring a diverse set of regions, slowing convergence while lessening vulnerability to local minima, with the converse true of prioritising the social term.

Different problems, and indeed different stages of solving the same problem, may require a particular balance of expansion and exploration. The constants c_1, c_2 are unsuitable for adjusting this balance as they also govern the convergence to previously discovered positions [51; 70]. In order to overcome this limitation, Shi and Eberhart in [66] added an **inertia weight** coefficient η to the previous velocity term in the velocity update equation, giving

$$\vec{v}_i(t) = \eta\ \vec{v}_i(t-1) + c_1\ \vec{r}_1\ \otimes\ (\vec{a}_i(t) - \vec{x}_i(t)) + c_2\ \vec{r}_2\ \otimes\ (\hat{h}(t) - \vec{x}_i(t)) \tag{2.8}$$

We can see that this allows us to explicitly control the balance between expansion and local search.

It can be advantageous to have different search behaviour over the run-time of the algorithm. Specifically it is wasteful to spend many function evaluations attempting to exploit local minima in currently discovered locations, if we have not already discovered a promising (low cost) region in the fitness space. Conversely, if we have

discovered a promising region it may require many additional function evaluations to find a better one. It is also impossible to properly compare two similarly promising regions in the search space without any local exploration in order to gain a good estimate of the true value of their respective minima.

Because the value of η allows us to control the focus of the algorithm's search behaviour, we can change its value when necessary to take account of the trade off between global exploration and local search. Although their are several methods for choosing the value of η [53], Eberhart and Shi [22] have shown that making η a linearly decreasing function of time (i.e. **annealing** η) permits the algorithm to find good regions early in its run time and then exploit them better as it nears its end, improving performance, particularly in multi-modal landscapes.

2.2.2.3 Constriction Coefficient

As an alternative to artificially clamping the velocity to provide stable particle trajectories, Clerc and Kennedy [15] proposed adding a **constriction coefficient** χ to the velocity update equation, so that it was of the form

$$\vec{v}_i(t) = \chi[\ \vec{v}_i(t-1) + c_1\ \vec{r}_1\ \otimes\ (\vec{a}_i(t) - \vec{x}_i(t)) + c_2\ \vec{r}_2\ \otimes\ (\hat{h}(t) - \vec{x}_i(t))] \qquad (2.9)$$

By analytically analysing PSO from the perspective of an arbitrary individual particle, they calculated the value of χ at the boundary between the set of values forcing convergence and those at which velocity could increase unbounded This value is given by

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \qquad (2.10)$$

for $\phi = c_1 + c_2 > 4$.

By setting the constriction coefficient to this critical value we retain the stability of velocity clamping. The advantage we gain, however, is that for a constriction coefficient at or below this value a particle is guaranteed to converge to some weighted combination of its local attractor and the global historic best, providing they remain stationary. As the trajectory of a converging particle is damped oscillation about the point it is converging to [51], the constriction coefficient guarantees us a local search, something which is not true of velocity clamping.

Note that as χ is merely a constant multiplicative factor running through the velocity update equation it does not need to be regarded as separate to the other

update methods discussed. Instead it can be regarded as a principled method for choosing η, c_1 and c_2 in order to get some desirable behaviour. Secondly, although it was originally proposed to replace velocity clamping, empirical studies [22] show that an algorithm using *both* velocity clamping, with $v_j^{max} = b_j^{max} - b_j^{min}$, and the constriction coefficient gives better performance than either do, individually.

2.2.2.4 Topology

As mentioned previously, the original PSO algorithm grew out of a simplified simulation of flocking [38]. In this precursor model, the agents' behaviour was driven by the velocities of their "neighbour" agents, those agents closest to them in the search space. As the algorithm was developed for optimization purposes, the system was simplified and the topological aspect removed meaning that all agents had access to information from the entire swarm.

Variants of PSO have since been designed in which this "social" sharing of information has been restricted to subsets of the swarm, or **neighbourhoods**. The nearest neighbour Euclidian approach taken originally was found to be too computationally expensive [53] and was abandoned. Instead, it is common in current variants to assign each particle i a static set of neighbours, $\mathcal{N}_i$ and limit the sharing of information to this subset. As, in the canonical PSO algorithm, global information sharing allows each particle to know the global historic best position of the swarm, we must modify this so that each particle instead knows a **local historic best** position, given by

$$\hat{h}_i(t) = \underset{\vec{h}_j(t)}{\operatorname{argmin}} f(\vec{h}_j(t)), \quad j \in \mathcal{N}_i \tag{2.11}$$

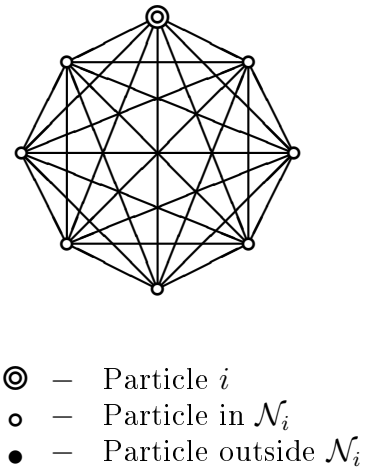
This is then used to update the velocity in exactly the same manner as in equation 2.8, giving

$$\vec{v}_i(t) = \eta\ \vec{v}_i(t-1) + c_1\ \vec{r}_1\ \otimes\ (\vec{a}_i(t) - \vec{x}_i(t)) + c_2\ \vec{r}_2\ \otimes\ (\hat{h}_i(t) - \vec{x}_i(t)) \tag{2.12}$$

Note that as this makes no modification to the overall structure of the velocity update equation, it is entirely compatible with the constriction factor shown in equation 2.9.

Clearly, how we set $\mathcal{N}_i$ will affect the overall behaviour of the algorithm and numerous topologies have been proposed. Here we discuss three popular variants.

Figure 2.1: Example fully connected topology for $n = 8$ and $i = 1$. We have $\mathcal{N}_1 = \{1, ..., 8\}$.



Fully Connected

The fully connected topology, an example of which is given in figure 2.1, is commonly known as the **gbest** version of the PSO algorithm. It defines the original PSO algorithm, in which all particles share their historic best positions with all other members of the swarm, in terms of a neighbourhood given by

$$\mathcal{N}_i = \{1, ..., n\} \tag{2.13}$$

Note that, using this topology, $\hat{h}_i(t) \equiv \hat{h}_j(t)$ for all $1 \leq i, j \leq n$ and so all particles in the swarm will be attracted to a single, low cost, point. As such it is believed that gbest-based algorithms should converge quickly but be more vulnerable to local minima [39].

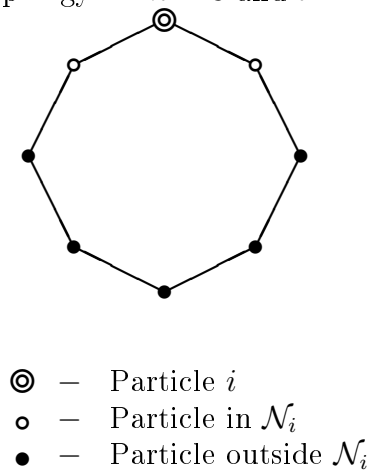
Ring

With the aim of making PSO more robust against the effect of local minima the ring topology, or **lbest** algorithm, was introduced [20]. Conceptually, the particles are placed in a ring and form a local neighbourhood with their neighbouring particles (see figure 2.2 for an illustration of this). More formally, given an enumeration of the particles we can define their neighbourhoods as

$$\mathcal{N}_i = \{i, i \pm 1 \bmod n\} \tag{2.14}$$

Note that, due to the convention of enumerations beginning with 1, we use the

Figure 2.2: Example ring topology for $n = 8$ and $i = 1$. We have $\mathcal{N}_1 = \{1, 2, 8\}$.



upper modulo function (i.e. taking $n \bmod n = n$). This is merely for notational convenience.

As this an example of a bi-directional neighbourhood[1] which has the smallest maximum neighbourhood size, we expect information about good positions to be spread slowly. This means that the swarm will be able to maintain searches in multiple locations, decreasing the effect of local minima, but is likely to converge more slowly than a fully connected swarm.

von Neumann

The von Neumann topology can be seen as a extension of the ring topology to two dimensions. We place the particles on an $a \times b$ grid (where $a, b > 2$ and $ab = n$). We then construct two ring sub-neighbourhoods for each particle, one along its row and one along its column. The particle's true neighbourhood is taken to be the union of these sub-neighbourhoods.

More formally, it is clear that we can construct a bijective function

$$f_{a,b} : \{1, ..., a\} \times \{1, ..., b\} \to \{1, ..., n\} \tag{2.15}$$

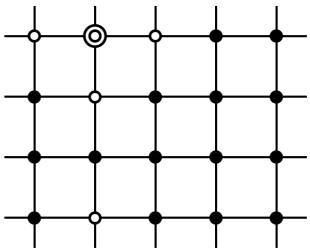
which sends any point on this grid to a unique particle. Let $f_{a,b}(u, v) = i$, then

$$\mathcal{N}_i = \{i, f_{a,b}(u \pm 1 \bmod a, v), f_{a,b}(u, v \pm 1 \bmod b)\} \tag{2.16}$$

Note that as search-space topology is not respected (i.e. a particle's position is independent of its index, i), the exact form of $f_{a,b}$ is unimportant. Studies have shown the

[1] Neighbourhood in which $j \in \mathcal{N}_i \Leftrightarrow i \in \mathcal{N}_j$.

Figure 2.3: Example von Neumann topology for $n = 20$, $a = 5$, $b = 4$ and $i = 3$. We have $\mathcal{N}_2 = \{1, 2, 3\} \cup \{2, 7, 17\} = \{1, 2, 3, 7, 17\}$.



◎ – Particle i
○ – Particle in $\mathcal{N}_i$
● – Particle outside $\mathcal{N}_i$

von Neumann topology to be a good compromise between the fully connected and the ring topologies, providing a good level of both robustness and convergence speed [39]. As particles are highly dynamic, and information spreads quickly through the grid, the initial choice of neighbourhood has a minimal effect on performance. In fact Mendes [47] shows that the particular topology chosen may not be crucial, finding that "*All topologies with degree 4 have consistently good proportion of successes*".

2.2.2.5 Noise-resistant PSO

Particles in standard PSO deterministically remember the lowest cost position that they have evaluated over the course of an algorithm run. This has two main drawbacks in a stochastic system:

- Noise in the region of a local minimum can render the true minimum difficult to find. If the noise is typically large enough to dominate the structure of the local search space, then we can have little confidence in a single evaluation of a given point.

- If a single extreme variation severely decreases the apparent cost of a given point, this can critically affect the performance of the entire algorithm. This single point could be taken as the neighbourhood historic best of a large number of particles in the swarm, attracting them to a single, possibly suboptimal, region of the search space.

As such, when optimising a stochastic system we need to ensure that our algorithm is robust with respect to stochasticity. Pugh et al. [58] have developed a simple variant of PSO in which, if a particle's historic best position is not replaced in a given iteration, its cost is re-evaluated and its true cost is taken to be the average of all previous values. As this variant outperforms standard PSO in the presence of noise, we use it when optimising stochastic systems.

2.2.3 Comprehensive Learning PSO

While PSO performs well on a variety of problems, it can be prone to premature convergence on the more difficult, multi-funnelled problems [67]. We say that a system has **converged** if all its particles are clustered around a single point and their velocities are such that they are incapable of escaping from this state. This has the advantage of facilitating a thorough local search, but comes at the cost of global search capacity. We say that it has undergone **premature convergence** if the likely cost reduction in continuing with a global search outweighs the advantages of fully optimising the current best minimum, given the constraints the algorithm is operating under (e.g. time, cost).

Note that equation 2.7 clarifies the cause of premature convergence. If there is some $\hat{h}$ which is sufficiently difficult to improve upon, it tends to remain fixed for long enough such that (assuming f is continuous) the swarm's attractors cluster around it. Once a particle i is oscillating about a single location, the magnitude of the velocity tends to decrease due to the contribution of the random vectors $\vec{r}_1$, $\vec{r}_2$, causing $\vec{a}_i$ to approach $\hat{h}$. Once all velocities are sufficiently small and all personal bests sufficiently close to the global best, the particles are trapped and all search capacity is lost. Using a topology with fewer connections slows the speed of this information spread, but does not stop it.

To guarantee that the swarm converges into such a small area, we require that $c_1 + c_2 < 4$ (see [51] for theoretical proof of this in a simplified system). It would, therefore, seem reasonable to increase the magnitude of these constants to prevent premature convergence. However, this not only prevents premature convergence, it prevents *any* convergence, meaning that the swarm cannot cluster around and efficiently exploit good locations in the search space. For this reason, other methods

of preventing premature convergence must be looked at. Therefore we look at a PSO variant designed to combat this problem, comprehensive learning PSO.

Comprehensive learning PSO (CLPSO) is a PSO variant introduced by Liang et al. [44] which outperforms many current PSO variants, particularly on difficult multimodal problems.. It attempts to slow convergence by allowing particles to generate **exemplar** positions, $\vec{e}_i(t)$, which they are attracted to. These exemplars are stochastic combinations of a particle's own knowlege and that of its neighbours. As exemplars are unique to a particle they allow swarms to retain their diversity. A CLPSO particle updates its velocity according to the following equation.

$$\vec{v}_i(t) = \eta\ \vec{v}_i(t-1) + \phi\ \vec{r}\ \otimes\ (\vec{e}_i(t) - \vec{x}_i(t)) \tag{2.17}$$

Particles generate a single exemplar, learn form it, and then discard it in favour of a new position. In an attempt to ensure that good exemplars are retained for long enough for a particles to exploit them fully, while bad exemplars are discarded quickly to minimise wasted function evaluations, the concept of particle stationarity is used. A particle i is said to have been **stationary** for τ time steps if τ is the largest number for which:

$$f(\vec{a}_i(t-s)) = f(\vec{a}_i(t)), \forall s < \tau. \tag{2.18}$$

A particle generates a new exemplar when it has been stationary for m timesteps, where m is called the **refreshing gap**.

When generating a new exemplar, a particle starts with its own attractor as a base position. Each dimension of this exemplar vector is then independently modified with probability Pc_i (or not modified with probability $1 - Pc_i$), called the **learning probability** for agent i. To modify dimension d of the exemplar, two distinct neighbours of the particle are chosen at random, and the one with lower cost $\vec{h}(t)$ is deterministically selected (i.e. if neighbour j is selected then $e_{i,d}(t+1)$ is set to $h_{j,d}(t)$). If this processes finishes with no modification to the exemplar, one dimension of is randomly selected for the insertion of a neighbouring value. See algorithm 1 for details.

Liang et al. [44] found that better results were obtained when particles had a heterogeneous set of learning probabilities and we will take their approach when using CLPSO, setting each particle's learning probability according to the following

equation.

$$Pc_i = 0.05 + 0.45 \cdot \frac{\exp\left(\frac{10 \cdot (i-1)}{n-1}\right) - 1}{\exp(10) - 1} \tag{2.19}$$

Algorithm 1 Exemplar Selection

```
for 1 ≤ j ≤ d do
  Randomly choose r ~ U[0, 1]
  if r < Pc_i then
    e_{i,j} = a_{i,j}
  else
    Randomly choose u, v ∈ N_i such that u, v and i are all distinct.
    if f(h_u) < f(h_v) then
      e_{i,j} = h_{u,j}
    else
      e_{i,j} = h_{v,j}
    end if
  end if
end for
if e_i ≡ a_i then
  Randomly choose j ∈ {1, ..., d}
  Randomly choose u, v ∈ N_i such that u, v and i are all distinct.
  if f(h_u) < f(h_v) then
    e_{i,j} = h_{u,j}
  else
    e_{i,j} = h_{v,j}
  end if
end if
```

3 Distributed Task Allocation

CONTENTS

3.1 Motivation

Distributed systems are an active and important field with applications ranging from distributed heterogeneous computing systems [32] to mobile sensor networks [45], and their performance is dependent on the coordination of disparate sub-systems to fulfil the overall goal of the system. This problem can be seen as one of efficiently deploying a finite set of resources in order to complete a distributed set of sub-tasks, where these sub-tasks further this overall goal. It is clear that, in theory, the best method for coordinating these resources must be centralised. A central controller can, at minimum, issue commands causing resources to be deployed as if according to the optimal decentralised behaviour. In fact, access to global information and the ability to coordinate agents should allow better performance than any collection of individuals. However, limitations on resources such as computational power and/or communication costs [36] mean that centralised solutions are not efficient in practice. This is particularly true for large systems as the calculation time of an optimal allocation of tasks becomes a major limitation [13]. Large systems also decrease the effectiveness of global inter-agent communication. Shehory et al. [65] point out that if n agents are communicating with each other, this involves a total of $O(n^2)$ communications, potentially "overwhelming" the communication network.

Rana et al. [59] suggest that many practical applications will require large numbers of agents, and because of this the poor scalability of centralised systems rules them out as solutions to these problems. Therefore, decentralised approaches must be investigated. In a decentralised approach, a set of autonomous, decision making, agents control the behaviour of sub-systems with the aim of coordinating to provide good global behaviour. A simple method to promote coordination would be to allow all agents to communicate freely, however n agents are communicating with each other involves a total of $O(n^2)$ communications [59]. If communication is costly, this is hardly an ideal approach and so finding good decentralised methods for distributed task allocation, which require a minimum of communication, is an important problem.

3.2 The Mail Processing Problem

The mail processing problem has been used in various comparative studies of algorithms for distributed task allocation, both presented as a problem of mail processing [27; 56; 57] and one of truck painting [11; 40]. It is a flexible problem which allows for a comprehensive investigation of the behaviour of candidate algorithms. The problem comprises a set of N_c mail producing cities, each of which is capable of producing and storing one batch each of N_m different mail types, and a set of N_a mail processing centres. Each centre has one associated mail collection agent whose task it is to travel to a city and return with a batch of mail for its centre to process.

We stress that the problem is intended as a prototype for general distributed task allocation rather than as a realistic model for either mail processing or truck painting. Although a fairly simple problem, it contains a number of characteristics useful for testing task allocation algorithms. It is:

- large scale, meaning that centralised methods are not practical.
- constrained, meaning that good trade-offs between various aspects of performance are necessary.
- flexible, allowing algorithms to be comprehensively tested .

Each mail type requires a different processing method and at any point in time the processing centre of agent a is specialised in one specific type σ'_a. When processing a batch of this mail type the centre can do so efficiently, taking a time t_p. However, in order to process a batch of a different mail type m, the centre must undergo alterations. This *c*hangeover $\sigma'_a \rightarrow m$ (including the processing of the batch) takes a time $t_c > t_p$.

In order to reduce the direct impact of these changeovers, each processing centre has a *m*ail queue in which it can temporarily store mail while processing other batches. This queue is capable of storing up to L_q batches of mail and, while there is space, the agent will continue to collect mail. A processing centre must process the mail in its queue in the order in which it arrives, such that all the freedom in the system is concentrated in the behaviour of the collection agents. Therefore, we define the *e*ffective specialisation σ_a of the agent as the last collected mail type, because σ'_a will be σ_a by the time the next collected mail is processed.

Because we wish to study algorithmic performance under settings requiring decentralised methods, we will use large number cities and a correspondingly large population of agents. In order to make this computationally viable we use the version of the problem studied in [27] in which we neglect city topology and information about this mail at a city is local and cannot be determined from outside the city. As this modified version of the problem is both large scale and only admits local information, it is ideal for studying the behaviour of decentralised, self-organising algorithms.

In order to simulate this system, we discretise time into steps of the amount of time it takes an agent to visit a city and return with mail. This allows us to define our measure of an algorithms performance, i.e. the *efficiency*, as the average amount of mail processed per agent per time step. During each time step the following happens:

1. Each centre processes the top batch of mail (if any), thus emptying one space in its queue, or proceeds with its changeover.

2. Each agent a picks one city c to visit, unless its processing centre has a full queue.

3. Each city now has a set Ψ_c of visiting agents and agents proceed to choose mail at these cities.

4. Then each agent returns to its processing centre and deposits the chosen batch (if any), into the queue.

5. Finally, the cities increase the stimuli of left-over batches and produce new batches.

Note that as a further constraint upon the agents we do not allow them to self-determine the order in which they see mail, or to retrospectively decide to take a batch of mail they have previously rejected even if no other agent has taken it. To avoid bias, mail is offered to agents in a random order either agent by agent, or as a group. These steps are described more formally in algorithm 2.

It is possible to consider many different schemes under which agents choose cities, but in this chapter we will only consider random city choice. The reasons for this

Algorithm 2 An iteration of the mail processing problem at time t.

for agent $a = 1, ...N_a$ **do**
  if queue isn't full (if $l_a \neq L_q$, where q_{l_a} is the last non-zero entry in $\boldsymbol{q}_a$) **then**
    Choose a city (add a to the set Ψ_c) for random $c \in \{1, ...N_c\}$
  end if
end for
for city $c = 1, ...N_c$ **do**
  for mail type $m = 1, ...N_m$ **do**
    if mail of type m is not already at city c (i.e. if $w_{c,m} = 0$) **then**
      produce mail of type m (set $w_{c,m} = 1$) with probability $\pi_m(t)$.
    end if
  end for
  Agents in Ψ_c act at city c, **details are algorithm dependent**
  for mail type $m = 1, ...N_m$ **do**
    if mail of type m is already at city c (i.e. if $w_{c,m} \neq 0$) **then**
      increase waiting time of mail type m (set $w_{c,m} = w_{c,m} + 1$).
    end if
  end for
end for
for agent $a = 1, ...N_a$ **do**
  if agent a accepted a piece of mail (type denoted m) **then**
    if mail matches effective specialisation (if $m = \sigma_a$) **then**
      add mail to the processing queue ($q_{a,l_a+1} = t_p$)
    else
      add mail to the processing queue with a penalty ($q_{a,l_a+1} = t_c$)
      switch effective specialisation to account for the changeover ($\sigma_a = m$)
    end if
  end if
  if agent a has mail in its queue (if $q_{a,1} \neq 0$) **then**
    Continue to process the first item in the queue
    if the agent has finished processing a batch of mail (if $q_{a,1} = 0$) **then**
      move remaining batches of mail up in the queue
      (set $q_{a,l} = q_{a,l+1}, l = 1, ..., L_q - 1$ and $q_{a,L_1} = 0$)
    end if
  end if
end for

choice are twofold. Firstly, for the generality of the problem. It is easy to imagine a real-world problem in which a systematic method of returning to task locations is

impractical. Task locations may be non static, rendering memories of past locations obsolete, or the environment may be difficult to navigate, making the problem of agent navigation more difficult than the initial problem of task allocation. Therefore, it is important to consider a version of this problem in which agents are incapable of preferential choice of task site. Secondly, current implementations of decentralised task allocation algorithms have not been designed or tested with other systems of city choice, and the introduction of such a system could bias comparative results.

No centralised control of the agents is permitted, such that the aim is to give agents a set of autonomous rules in order to maximise the amount of processed mail. We refer to this in terms of average mail processed per agent per time step, or *efficiency*. It is clear that this efficiency is limited by several factors. The agent must strike a balance between maximising the proportion of times it takes mail when it visits a city and minimising the amount of time which it spends not visiting cities due to its centre having a full queue. The likelihood of having a full queue increases with the agents likelihood to take mail of a type different from that it took previously. In an ideal situation, an agent would always take mail of the type that it took previously. However, this ideal situation is impossible unless the agent rejects all types of mail other than the one previously taken, which in turn is sub-optimal as it increases the number of times the agent returns empty handed and in extreme cases may even lead to a deadlock situation in which all mail of a certain type is rejected by all agents.

Each city c has a vector $\boldsymbol{w}_c = (w_{c,1}, .., w_{c,N_m})$ where $w_{c,m}$ is the waiting time of the batch of mail type m. A piece of mail with high waiting time is representative of a high priority task. Note that $w_{c,m} = 0$ indicates that there is no batch of that type of mail present, either because no such batch was produced, or because another agent has already taken that batch. Upon production of a batch of mail type m, its waiting time is initialised to $w_m = 1$, and at the end of each time step the waiting times of remaining batches of mail are increased by 1.

For each agent a, each mail type represents a distinct task in which it engages upon uptake of that batch. Agents visit cities and, once there act in the manner determined by their algorithms. Specifically, for a threshold based algorithm agents are individually allowed to examine mail at a city, accepting or rejecting each batch in turn based on their response thresholds. For market based algorithms *all* agents

at a city are offered each piece of mail in turn and must submit a bid determined by their bidding function, with the highest bidder taking the batch of mail. Once an agent selects a piece of mail, it is then deposited in the queue of the agent's processing centre, $\boldsymbol{q}_a = (q_{a,1}, .., q_{a,L_q})$, which can store a backlog of up to L_q batches of mail.

Two different types of environment are considered:

- A static environment, in which a city automatically replaces each taken batch of mail at the end of each time step.
- A dynamic environment, in which the probabilities of production of batches of the mail types varies over time.

The dynamic environment is specifically designed to test the flexibility of the system with respect to continuous changes. We have chosen to vary the probability of taken mail batches being replaced in a sinusoidal fashion, although the exact form of the function is not critical. All mail types have the same wavelength (e.g. to mimic seasonal variations), but have a different phase. Hence, all mail types have periods of both high and low production with certain mail types being dominant at some times and scarce at others. The probability of creating a taken batch of type m at the end of cycle t is given by:

$$\pi_m(t) = \begin{cases} 1, & \text{static} \\ \frac{1}{2}[1 + \sin(\frac{t2\pi}{\xi} - \frac{m2\pi}{N_m})], & \text{dynamic} \end{cases} \tag{3.1}$$

where ξ is the wavelength. In the dynamic environment an agent may be forced to compromise in its strategy of specialising in one type of mail as there will be periods during which its preferred mail type is rare. Note that as mail is produced independently for each mail type m and at each time step t, $\pi_m(t)$ is a well defined probability.

In addition to environmental changes, we also investigate the robustness of the system under abrupt changes. To this purpose, we consider the case where all agents specialised in a particular mail type are suddenly removed, and monitor the ability of the system to adapt quickly to the change. This represents one of the most catastrophic failures in terms of instantaneous loss of efficiency, and should give an fair indication of the general robustness.

It is clear that the problem is to minimise the loss of efficiency. Therefore, it is important to identify the different mechanisms that lead to efficiency loss. If an agent of effective specialisation σ_a fails to process mail during an iteration, this can be categorised into four cases:

($\boldsymbol{\ell}$.1) The agent is inactive due to a full mail queue at its processing centre.

($\boldsymbol{\ell}$.2) Mail type σ_a is available at the city, but the agent rejects all mail.

($\boldsymbol{\ell}$.3) Mail type σ_a is unavailable at the city and the agent rejects all mail.

($\boldsymbol{\ell}$.4) There is no mail at all available by the time of the agent's action.

While it is possible to minimise $\boldsymbol{\ell}$.4 by increasing $\boldsymbol{\ell}$.1-$\boldsymbol{\ell}$.3 (i.e. by lowering the overall acceptance rate of mail), this is clearly not ideal. In particular, $\boldsymbol{\ell}$.3 and $\boldsymbol{\ell}$.4 are due to the non-uniform number of agents visiting cities. In the current problem we have no control over this and focus mainly on the other sources. Clearly $\boldsymbol{\ell}$.2 is unnecessary, as the uptake of mail of its own specialisation has no negative consequences for an agent.

One should note that $\boldsymbol{\ell}$.1 and $\boldsymbol{\ell}$.3 are finely balanced against each other as a greater uptake of mail of non-specialised types leads to an increase in agents with full queues. This relationship is clearly non-linear as a increase in the number of inactive agents leads to an increase in average stimulus, hence to an increase in the uptake of non-specialist mail and an even further increase in the number of inactive agents.

As market based algorithms never reject mail, they are unaffected by $\boldsymbol{\ell}$.2 and $\boldsymbol{\ell}$.3. This means that they are never affected by the unnecessary $\boldsymbol{\ell}$.2, but are also unable to actively balance $\boldsymbol{\ell}$.1 against $\boldsymbol{\ell}$.3.

As the only freedom lies in the behaviour of agents at cities, one can only minimise these loss sources by selecting an agent rule-set which has the optimal balance between loss sources under the given circumstances. In particular, we wish for agents to take mail of the same type on consecutive occasions with the high probability, minimising changeovers and the probability of their queues filling up. However, agents should retain some flexibility and should not compromise their ability to adjust to the current level of demand in the system (giving a long term advantage) to avoid a single changeover (a short term penalty).

To illustrate this, it is possible to imagine two rules for agent behaviour designed to highlight the extremes of possible algorithms. In the first, agents would "greedily" take mail if it was available ensuring cities' were maximally served in the short term. In the second, agents would be completely "selective", refusing to take any mail type other than the one they were specialised in. Clearly the first approach has no method of reducing $\ell.1$ and the second, no method to reduce $\ell.3$. Thus, a good algorithm must involve some compromise between greed and selectivity.

3.3 Threshold Based Algorithms

The behaviour of social insects has been a good source of inspiration in the design of multi-agent-systems (MAS) [9]. In particular, we are interested in algorithms using the principle of "stigmergy", introduced by Grassé [28]. Stigmergic mechanisms are ones in which agents coordinate themselves using the environment rather than through direct communication. Behaviours, which are triggered by observation of the environment, also cause environmental change, thus modifying the behaviour of other agents. This coordination without communication makes stigmergy a very attractive paradigm when designing MAS.

Therefore, our first candidate solutions to the mail processing problem are inspired by natural systems, and are based on the variable response threshold model of task allocation [8; 68]. This model is a stigmergic mechanism developed to explain division of labour in social insect colonies. Price [56]; Price and Tiňo [57] show that agents using rules based on the threshold model exhibit good performance on this problem when compared with several other algorithms, particularly when flexibility is required. Algorithms based on this model have also been successfully applied to other problems which require robust, decentralised control including the scheduling of truck painting [11; 40; 50], which involves similar constraints, and the real world example of conserving battery life in a remote sensor network [36].

The idea of thresholds as a method for task allocation was developed by [8] in order to show how the flexibility of insect colonies to different circumstances can be explained by the autonomous flexibility to engage in tasks of the individuals which comprise them. They proposed a model, known as the fixed response threshold (FRT) model, which stated that the tasks that an individual is capable of engaging

in could be broken down into types and that each instance of a task has some stimulus associated with it which is indicative of its demand for completion. Each colony member has a set of thresholds which determine their preference for engaging in each type of task. Upon encountering a task an individual will compare the stimulus s of the task with the corresponding threshold θ, and use this to determine the probability of uptake of that task. The probability of uptake should be high for $s >> \theta$, low for $s << \theta$, zero for $s = 0$ (no demand for the task), and $\frac{1}{2}$ for $s = \theta$, and is defined by a *threshold function* $\Theta(s, \theta)$.

In the original formulation of the model it was assumed that each agent had a genetically determined set of thresholds which were constant over an agent's lifetime. However, while this was able to account for such features of social insect colonies as increased uptake of tasks by non-specialist individuals upon the removal of specialists in these tasks, it was unable to account for the initial distribution of specialisations or the re-specialisation of colony members to meet changes in the distribution of stimulus. In order to account for these features [68], introduced a process of self reinforcement whereby time spent performing a task would lead to a decrease in an agent's threshold for this task whilst time spent not performing this task would cause the threshold to increase. Individuals possess an update rule, U, which governs the magnitude of these changes. This model is known as the variable response threshold (VRT) model. In this chapter a discretised version of this model is used and increases in thresholds at times when the individual is inactive are discounted. As such an agent with thresholds $\vec{\theta} = (\theta_1, ...\theta_n)$ will, upon completion of a task of type i, update its thresholds as follows:

$$U(\boldsymbol{\theta}, i) = (u(\theta_1, i), ..., u(\theta_n, i)) \tag{3.2}$$

where

$$\begin{cases} u(\theta_j, i) < \theta_j & \text{if } i = j, \\ u(\theta_j, i) > \theta_j & \text{otherwise,} \end{cases} \tag{3.3}$$

with θ_j restricted to some range $[\theta_{min}, \theta_{max}]$. These changes are related to both the size of the threshold and the time taken to complete the task.

We introduce a set of new update rules and analyse their behaviour when applied to a general model of distributed task allocation, the "mail processing" problem. We use these rules as "species" in an evolution strategies algorithm to determine the best

rule in a given circumstance and to optimise the parameters of these rules. These are statistically analysed from a population dynamics perspective. The flexibility of the system is also tested in a dynamic environment in which task production probabilities are continuously varied, and with respect to a catastrophic breakdown in which all agents specialised in a particular task cease functioning.

3.3.1 Mail Uptake

Threshold based algorithms can be seen as an advance on the selective algorithm outlined at the end of section 3.2. Each agent has a set of thresholds related to its desire to engage in particular tasks. In this case, as tasks types are represented by mail types, each agent a has a set of thresholds $\boldsymbol{\theta}_a = (\theta_{a,1}, .., \theta_{a,N_m})$, where $\theta_{a,m}$ is the agent's threshold for mail type m. This threshold is related to the *stimulus* s of a particular task instance, indicating the demand of a task for completion using a threshold function, Θ which gives the probability of engaging in a task given a stimulus and corresponding threshold. The probability of engagement should be high for $s >> \theta$, low for $s << \theta$, zero for $s = 0$ (no demand for the task), and $\frac{1}{2}$ for $s = \theta$. Thus, for appropriate thresholds, agents will behave selectively in most cases but, for extreme stimulus, are capable of adjusting their behaviour.

Here we have chosen to use the exponential threshold function (ETF), which is the standard threshold function as proposed by Bonabeau et al. [8], and is defined, for threshold θ and stimulus s, as:

$$\Theta(s, \theta) = \begin{cases} \frac{s^\lambda}{s^\lambda + \theta^\lambda} & \text{if } s \neq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

which for $\lambda \geq 1$ has all the desired properties. Note that we enforce $\Theta(0, \theta) = 0$ (i.e. the task is not performed if there is no demand) to prevent undefined values when $s = \theta = 0$. In this problem, the stimulus is taking to be the waiting time of a batch of mail, which is a fairly natural measure of its demand for completion. This means that if an agent a encounters a batch of mail of type m at city c, its probability of accepting the mail is given by $\Theta(w_{c,m}, \theta_{a,m})$.

When a set of agents are visiting a city they are permitted to act, one at a time, in a random order. The agent which is currently acting then examines each piece of mail at a city again, one at a time, in a random order, with the probability

of mail uptake given by its threshold function. The agent continues either until it has accepted a batch of mail, or until it has rejected all batches. While it is clear that, for a given threshold function, the action of an agent at a city (and its short term efficiency) critically depends on its thresholds, its flexibility to adapt to new situations (and therefore its long term efficiency) critically depends on its ability to modify theses thresholds.

To adjust their behaviour to the state of the system, agents update their thresholds through self reinforcement using a strategy determined by an $u$pdate rule. After taking mail of type m, an agent replaces its thresholds with a set given by $U(\boldsymbol{\theta}, m) = (u(\theta_1, m), ..., u(\theta_{N_m}, m))$, where $u(\theta_m, m) < \theta_m$ and $u(\theta_j, m) > \theta_j$, $\forall j \neq m$. Thus, agents increase the chance of taking type m (decreasing $\boldsymbol{\ell}.2$) mail and decrease the chance of taking other mail types (decreasing $\boldsymbol{\ell}.1$). It is clear that a good update rule should drive an agent's thresholds to a state in which θ_{a,σ_a} is very low, thus avoiding $\boldsymbol{\ell}.2$ altogether.

3.3.2 Specialisation Update Rules

For a given threshold function, the efficiency of an agent critically depends on its thresholds, while its flexibility to adapt to new situations critically depends on its ability to modify its thresholds or *update rule*. The original threshold based algorithm, applied to the current problem in [56; 57], was proposed by Theraulaz et al. [68] as a model for social insect behaviour. It is possible, therefore, that this rule contains artefacts which are desirable to social insects, but which do not translate to optimal behaviour in a general setting.

The main goals of this section are to investigate what kind of threshold based rule is best suited to the problem at hand, and to investigate whether optimal update rules can be found autonomously by competition between the agents. Therefore, we compare the performance of some existing and some newly introduced update rules. We now proceed with a short overview.

The **Variable Response Threshold (VRT)** is the original nature inspired rule, proposed by Theraulaz et al. [68] and applied by Price [56]; Price and Tiňo [57].

The change in threshold $\Delta\theta_m$ over a period of time t, is given by:

$$\Delta\theta_m = -\epsilon\Delta t_m + \psi(t - \Delta t_m) \tag{3.5}$$

where Δt_m is the time spent performing task m, where $\epsilon,\ \psi$ are positive constants, and where θ_m is restricted to the interval $[\theta_{min},\theta_{max}]$. For the current model, (3.5) can be discretised, taking into account the fact that thresholds are only changed when a task is performed. Therefore, when the update rule is called, over a single time step, t will be 1 and Δt_m is 1 if mail type m was taken and 0 otherwise. Hence, the VRT rule can be rewritten as

$$u(\theta_m, i) = \begin{cases} \theta_m - \epsilon & \text{if } i = m, \\ \theta_m + \psi & \text{otherwise.} \end{cases} \tag{3.6}$$

A drawback of the VRT rule is that, in the event of a changeover, for small ϵ and ψ agents are unlikely to change their thresholds enough to have a high chance of picking the new mail type in the next time step, thus increasing $\boldsymbol{\ell}.2$ and potentially $\boldsymbol{\ell}.1$.

In order to overcome these flaws and to see if better efficiency could be obtained, we introduce another update rule.

The **Switch-Over (SO)** update rule, introduced by Goldingay and van Mourik [26], updates thresholds in a very simple manner: by fully specialising in the most recently taken mail type, and fully de-specialising in all other mail types:

$$u(\theta_m, i) = \begin{cases} \theta_{min} & \text{if } i = m, \\ \theta_{max} & \text{otherwise.} \end{cases} \tag{3.7}$$

In some sense the switch-over rule can be seen as an extreme case of the VRT rule with $\epsilon,\ \psi \geq \theta_{max} - \theta_{min}$. Such values, however, are not in the spirit of the VRT, and so it makes sense to consider them as separate cases. We introduce SO as we expect it to minimise $\boldsymbol{\ell}.1$ and $\boldsymbol{\ell}.2$ in a static environment, while a drawback is that $\boldsymbol{\ell}.3$ could be maximised.

The **Distance Halving (DH)** rule, keeps the thresholds in the appropriate range by halving the Euclidean distance between the current threshold and the appropriate

limit.

$$u(\theta_m, i) = \begin{cases} \frac{\theta_m + \theta_{min}}{2} & \text{if } i = m, \\ \frac{\theta_m + \theta_{max}}{2} & \text{otherwise.} \end{cases} \tag{3.8}$$

Note that DH takes several time steps to fully specialise, but effectively de-specialises in a single time step.

The **(Modified) Hyperbolic Tangent ((m)tanh)** rule is introduced in a similar spirit to the VRT rule, allowing continuous variation of the thresholds without artificial limits. The thresholds are a function of some hidden variables h_m:

$$\theta_m = \theta_{min} + (\theta_{max} - \theta_{min}) \frac{1}{2}(1 + \tanh(h_m)), \tag{3.9}$$

Note that any sigmoid function could replace tanh(). The update rule works on the h_m similarly to the VRT (i.e. $h_m^{new} = u'(h_m^{old}, i)$):

$$u'(h_m, i) = \begin{cases} h_m - \alpha & \text{if } i = m, \\ h_m + \beta & \text{otherwise,} \end{cases} \tag{3.10}$$

for some positive constants α and β. As for the VRT, low α and β values may suffer from slow re-specialisation, however, the speed of re-specialisation is independent of θ_{min} and θ_{max}. Furthermore, the rule may lead to the problem of saturation. For large hidden variables, the tanh rule produces insignificant changes to the actual thresholds. As the whole basis of VRT-like models is that engagement in a task leads to an increased likelihood of repeating the task, this problem must be addressed. Therefore, we modify the tanh rule by making a distinction between positive and negative hidden values, and introduce re-specialisation coefficient $\eta \in [-1, 1]$ such that

$$u'(h_m, i) = \begin{cases} h_m - \alpha & \text{if } i = m \text{ and } h_m \leq 0, \\ \eta\ h_m - \alpha & \text{if } i = m \text{ and } h_m > 0, \\ h_m + \beta & \text{if } i \neq m \text{ and } h_m \geq 0, \\ \eta\ h_m + \beta & \text{otherwise.} \end{cases} \tag{3.11}$$

It is identical to the tanh rule for $\eta = 1$, but is more similar to the SO rule for η close to -1.

For ease of notation, we will refer to the VRT (resp. SO, DH, (m)tanh) algorithm rather that "a threshold based algorithm using the VRT (resp. SO, DH, (m)tanh) update rule".

3.3.3 Evolution Strategies

As we wish to study the absolute performance of our update rules in a manner unbiased by our original parameter choices, we optimise our parameter sets. Evolutionary algorithms seem a natural way of optimising the model, as it is inspired by the behaviour of social insects, and have been used before to good effect in similar settings (e.g. genetic algorithms in [10], [11]). We choose to use an evolution strategies (ES) algorithm [4] as its real valued encoding fits well with our problem. ES also allows for inter-update-rule competition, which enables us to find an optimal rule-set while we optimise the parameters rather than optimising each rule-set and choosing the best one. It is also possible that a population of different update rules can outperform a homogeneous population.

There are two peculiarities to note in our use of ES. Firstly, we are interested in looking at a range of update rules or "species" (possibly with different parameters spaces) within the same evolving population. As such, at initialisation we define each agent's object parameters to be a random update rule and its necessary parameters. This causes problems within ES as neither mutation to, nor recombination between, different species are well defined. As such we do not allow inter-species mutation and have enforced $\rho = 1$.

Secondly, the performance of an individual agent is highly stochastic and dependent on the behaviour of other agents. In fact, our true fitness function is of the form $f(\mathbf{x}, \mathbf{P})$, where $\mathbf{P}$ is some population of agents, and is taken to be the average efficiency of the agent with parameters $\mathbf{x}$ over the course of a run amongst this population. Good agents in early (poorly optimised) populations are likely to have extremely high fitness as competition for mail is scarce. If we allow them to retain their fitness over many generations, this will strongly bias the ES towards their parameters. For this reason we evaluate the fitness of parents and children every generation in a population $\mathcal{P} \cup \mathcal{O}$.

3.3.4 A Theoretical Description

In general, the mail retrieval problem is hard to solve exactly as it depends on continuous variables (the thresholds), such that the number of micro-states of the agents is infinite (not countable). In other contexts where this problem occurs, such as continuous models on sparse random graphs (see e.g. [41] and references therein), population dynamics can be used for the theoretical analysis. For agent based models, however, this is tantamount to simulating the model. On the other hand agent based models are theoretically solvable when the number of discrete micro-states is finite. It turns out that we can analyse some non-trivial cases of the current model exactly.

For the SO algorithm, specialised agents only have thresholds in $\{\theta_{min}, \theta_{max}\}$, and thresholds are entirely determined by the effective specialisation. Therefore, a micro-state $\mathcal{A}$ of an agent is determined by the thresholds $\boldsymbol{\theta}$ associated with its specialisation σ, and the state $\boldsymbol{q}_L$ of the mail queue at its processing centre:

$$\mathcal{A} \equiv (\boldsymbol{\theta}, \boldsymbol{q}_L) \tag{3.12}$$

where $L \in \{0, .., L_q\}$ is the length of the queue and $\boldsymbol{q}_L = (q_1, .., q_L)$ are the remaining processing times. Note that if an agent is specialised in mail type m then $\theta_m = \theta_{min}$ while all other $\theta_n = \theta_{max}$, and that $q_1 \in \{1, .., t_c\}$ and $q_i \in \{t_p, t_c\}$ $(i > 1)$, as their processing has not yet started. Hence, the set $\mathcal{S}_\mathcal{A}$ of all possible agent micro-states has cardinality $|\mathcal{S}_\mathcal{A}| = N_m \ (1 + t_c \sum_{L=1}^{L_q} 2^{L-1}) = N_m \ (1 + t_c(2^{L_q} - 1))$.

The micro-states $\mathcal{C}$ of the city are already discretised, and consist of the waiting times of the mail types:

$$\mathcal{C} \equiv \boldsymbol{w}_{N_m} = (w_1, .., w_{N_m}) \tag{3.13}$$

Although the number of such states is infinite (waiting times have no upper limit), this is not a problem when the threshold function is such that $\Theta(w, \theta) = 1, \ \forall w > \theta_{max}$. Only states in $\{0, .., \theta_{max}, (>\theta_{max})\}^{N_m}$ need to be considered, and the set $\mathcal{S}_\mathcal{C}$ of all possible city micro-states has cardinality $|\mathcal{S}_\mathcal{C}| = (\theta_{max} + 2)^{N_m}$. For the threshold functions that we have considered, this is the case for sufficiently high λ because

then:

$$\Theta(w,\theta) = \begin{cases} 0 & \text{if } w < \theta \text{ or } w = 0, \\ 0.5 & \text{if } w = \theta \text{ and } w \neq 0 \ , \\ 1 & \text{if } w > \theta. \end{cases} \tag{3.14}$$

Defining the states of the agents as $\boldsymbol{s}(t) = \{s_a(t), a = 1, .., N_a\}$ and the states of the cities as $\boldsymbol{S}(t) = \{S_c(t), c = 1, .., N_c\}$, at any time t the global state of the system is completely determined by the agent profile $\boldsymbol{\mu}(t) = \{\mu_{\mathcal{A}}(t),\ \mathcal{A} \in \mathcal{S}_{\mathcal{A}}\}$ and city profile $\boldsymbol{\eta}(t) = \{\eta_{\mathcal{C}}(t),\ \mathcal{C} \in \mathcal{S}_{\mathcal{C}}\}$, where

$$\mu_{\mathcal{A}}(t) \equiv \frac{1}{N_a} \sum_{a=1}^{N_a} \delta_{s_a(t),\mathcal{A}} \ , \qquad \eta_{\mathcal{C}}(t) \equiv \frac{1}{N_c} \sum_{c=1}^{N_c} \delta_{S_c(t),\mathcal{C}} \ . \tag{3.15}$$

This is because, as agents and cities have no relevant features other than their micro-states $\mathcal{A}$ and $\mathcal{C}$, the system only depends on the proportion of agents and cities in each micro-state. Therefore the matrices $\boldsymbol{\mu}(t)$ and $\boldsymbol{\eta}(t)$, containing the density of each micro-state, contain all information about the system at time t.

In the large system limit, as a consequence of the Central Limit Theorem, $\boldsymbol{\mu}(t)$ and $\boldsymbol{\eta}(t)$ become deterministic quantities, for which we can derive the exact time evolution. It is convenient to break up the time evolution into four distinct steps:

a.1 changes to the $\boldsymbol{\mu}$ during mail uptake.

c.1 changes to the $\boldsymbol{\eta}$ during mail uptake.

a.2 changes to the $\boldsymbol{\mu}$ during processing of the queue.

c.2 changes to the $\boldsymbol{\eta}$ during mail production.

During the mail uptake the change in the agent profile can be described by multiplication with a matrix $\mathbf{T}(\boldsymbol{\mu}(t), \boldsymbol{\eta}(t))$ which explicitly depends on both $\boldsymbol{\mu}(t)$ and $\boldsymbol{\eta}(t)$ due to the competition between agents at the cities. The change to the agent profile during the processing of the queue, can be described by multiplication with a constant matrix $\mathbf{Q}$. The change in city profile during mail uptake can be described by multiplication with a matrix $\mathbf{L}(\boldsymbol{\mu}(t))$, which explicitly depends on $\boldsymbol{\mu}(t)$ due to competition between the agents. Finally the change in city profile during mail production can be described by multiplication with a matrix $\mathbf{P}(t)$ which is

time dependent for the dynamic environment only. Combined, this leads to the following exact time evolution:

$$\begin{cases} \boldsymbol{\mu}(t+1) & = \mathbf{Q} \quad \mathbf{T}(\boldsymbol{\mu}(t), \boldsymbol{\eta}(t)) \; \boldsymbol{\mu}(t) \; , \\ \boldsymbol{\eta}(t+1) & = \mathbf{P}(t) \; \mathbf{L}(\boldsymbol{\mu}(t)) \qquad \boldsymbol{\eta}(t) \; . \end{cases} \tag{3.16}$$

The derivation and exact expressions of the matrices $\mathbf{T}$, $\mathbf{Q}$, $\mathbf{L}$ and $\mathbf{P}$ are rather involved, and can be found in appendix A.1. The theoretical time evolution and numerical simulations are compared in the following subsection, and are in excellent agreement.

3.3.5 Results

In this subsection, we discuss the numerical results of our investigation into threshold based rules. First we describe the general tendencies, presenting results representative of our update rules in the static and dynamic environments. We also test the rules' robustness to sudden change with the removal of all agents specialised in a particular mail type. The second part of this section is dedicated to the optimisation of the parameters, and the selection of the best possible combination of update rules in terms of the overall efficiency. Thirdly, we compare our simulated results with the theoretical predictions made previously.

We investigate the performance of the different rules, both in terms of their behaviour under different conditions and in terms of absolute efficiency. However, investigating the influence of the system parameters on performance is beyond the scope of this chapter. Therefore, we have fixed the parameters to a *standard setting*. See appendix B.1.

3.3.5.1 General Tendencies

In order to investigate the dependence of our results on the system size, we vary N_a, while keeping N_m and $R_{a/c}$ fixed. In general, we find that the average of any measured quantity quickly ($N_a \simeq 5 \cdot 10^2$) converges to its asymptotic value (for $N_a = \infty$) with increasing system size, while both inter- and intra-run variance decreases to below the line width of the plots at values $N_a \simeq 10^4$. With these

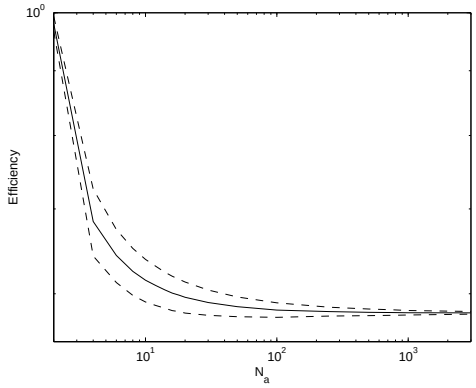


Figure 3.1: The average efficiency (solid line) of the SO algorithm as a function of the system size N_a with fixed $R_{a/m} = 1$ and $N_m = 2$, in a static environment on a log-log scale. Runs were averaged over $2 \times 10^5/N_a$ runs with error bars of $\pm$1s.d. (dashed lines).

findings in mind, we have decided to fix the system size at $N_a = 5 \cdot 10^4$, for which simulations can be run in reasonable time, and for which a single run suffices to determine any quantity with sufficient accuracy, omitting the need for error-bars in most figures that follow.

The significant finite size effects for relatively small system sizes are illustrated in figure 3.1 (which is representative for all threshold-based algorithms). The increased average efficiency for small values of N_a can be explained by considering that each agent on average competes with $(N_a - 1)/N_c$ agents, which is monotonically increasing with N_a for fixed $R_{a/m}$ and N_m and saturates for high N_a values.

As a rule of thumb, we consider an agent to be fully specialised in a mail type if its threshold for this type is less than a distance of 1% of the possible range from θ_{min} while all other thresholds are within 1% of θ_{max}. The qualitative behaviour of the SO algorithm, as shown in figure 3.2 (top), is typical for all update rules although the speed of convergence and asymptotic values, depend on both the update rule and threshold function. We see that the algorithm accounts for the genesis of specialisation. The system tends towards a stable asymptotic regime in which most agents are specialised and the specialists are equally split between mail types. We see that $\boldsymbol{\ell}.1$ is almost negligible while we have non-zero $\boldsymbol{\ell}.2$ is indicative of the high value of θ_{max}. With the S0 algorithm and $\theta_{min} = 0$, $\boldsymbol{\ell}.2$ becomes impossible once an agent has taken a piece of mail. Agents with all initial thresholds close to θ_{max} may never, over the course of a run, encounter a batch of mail with a strong enough

stimulus to accept it.

In the dynamic environment (see figure 3.2, bottom), we observe that the efficiency fluctuates about some average value over the course of a wavelength. The qualitative behaviour, as shown in figure 3.2 for the SO algorithm, is typical for most update rules and the behaviour of their average values is qualitatively similar to that in the static environment. However, the specialisation that drives this behaviour varies between rules. In particular, all rules based on hidden variables ((m)tanh) tend to specialise in a manner similar to the mtanh algorithm, while the other algorithms (VRT, SO and DH) behave qualitatively similarly to the SO algorithm.

The tanh function, like any sigmoid function, is effectively constant (i.e. saturated) for sufficiently large arguments. The saturation region is reached when an agent using the tanh update rule repeatedly takes the same mail type. Once in this region, the update rule becomes incapable of effective self reinforcement on which the VRT model relies, and incapable of reacting to changes in the environment. A similar problem can be encountered in neural networks with sigmoidal nodes, in which Hebbian learning drives synaptic weights into the saturation region of the function rendering the relative sizes of these weights meaningless [64] thus removing the selectivity of the node.

The tanh algorithm is the only algorithm inherently unable to dynamically adapt its thresholds due to the saturation effects described above, while the other algorithms can do so if given suitable parameters (i.e. a lowering of η in the mtanh algorithm). To highlight the effects of saturation for the tanh algorithm, we let the system equilibrate for 1500 iterations in the standard static environment to allow agents to specialise fully. Then we remove that half of the population that is most specialised in e.g. mail type 2, and equilibrate the remaining system (with halved $R_{a/m}$) for a further 1500 iterations. The results, shown in figure 3.3, show that in contrast to the SO algorithm, which adapts very quickly to the change, none of the specialised agents using the tanh algorithm re-specialise.

Stability is reached when the average stimulus of mail type 2 reaches a high enough level to force changeovers a significant proportion of the time, leading to high levels of $\boldsymbol{\ell}.1$. The mtanh algorithm is able to somewhat adapt to this by lowering the thresholds of agents taking type 2 mail most often, causing some to re-

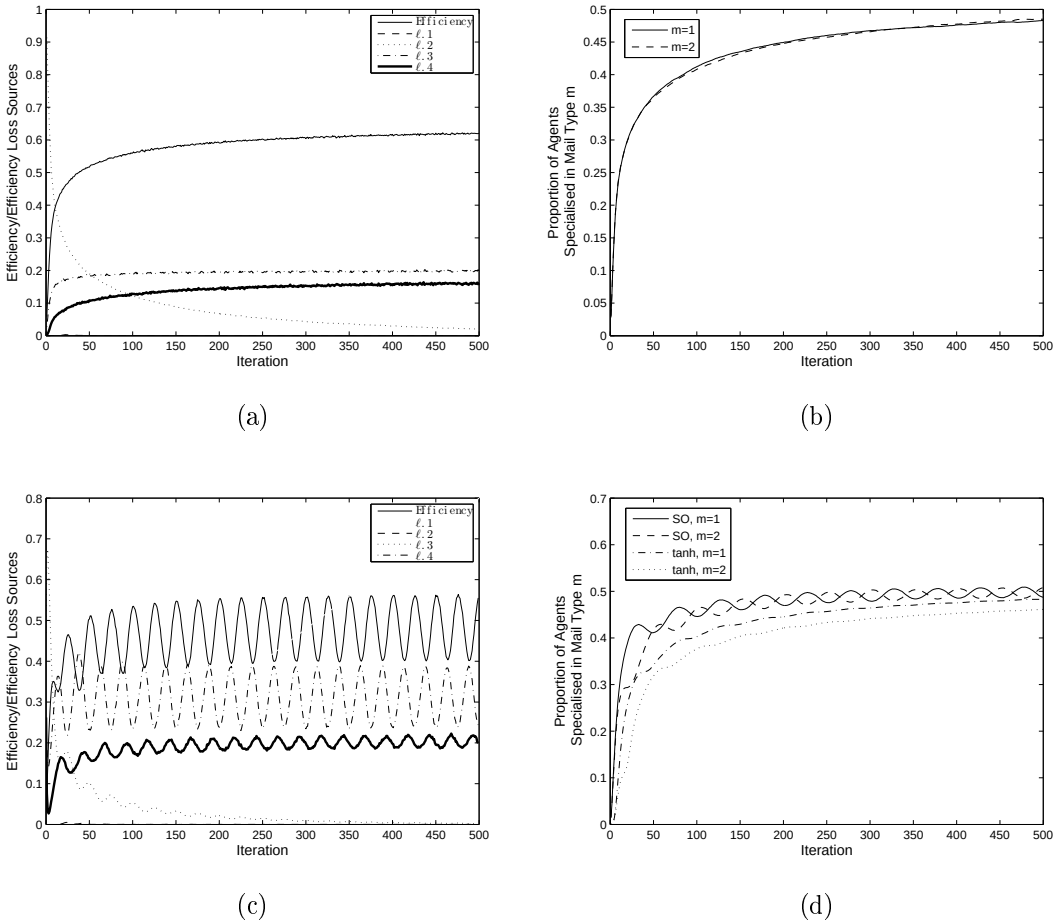


Figure 3.2: (a): evolution of the efficiency and loss sources during a single run in the standard static environment using the SO algorithm. Note that $\boldsymbol{\ell}.1$ is negligible everywhere and $\boldsymbol{\ell}.2$ tends to 0, while $\boldsymbol{\ell}.3$ and $\boldsymbol{\ell}.4$ (and hence the efficiency) quickly tend to their long time values. (b): the population of agents tends towards an equal split in specialisation with almost all agents specialised. (c): evolution of the efficiency and loss sources during a single run in the standard dynamic environment using the SO algorithm. The values of the loss sources and the efficiency fluctuate around their *average* values, which are qualitatively similar to those in the static environment. (d): the difference in specialisation behaviour between the SO and the tanh algorithms. Note that the tanh algorithm tends to a static, uneven (initial condition dependent) set of specialisations, while the SO algorithm efficiently adapts to changes in the environment.

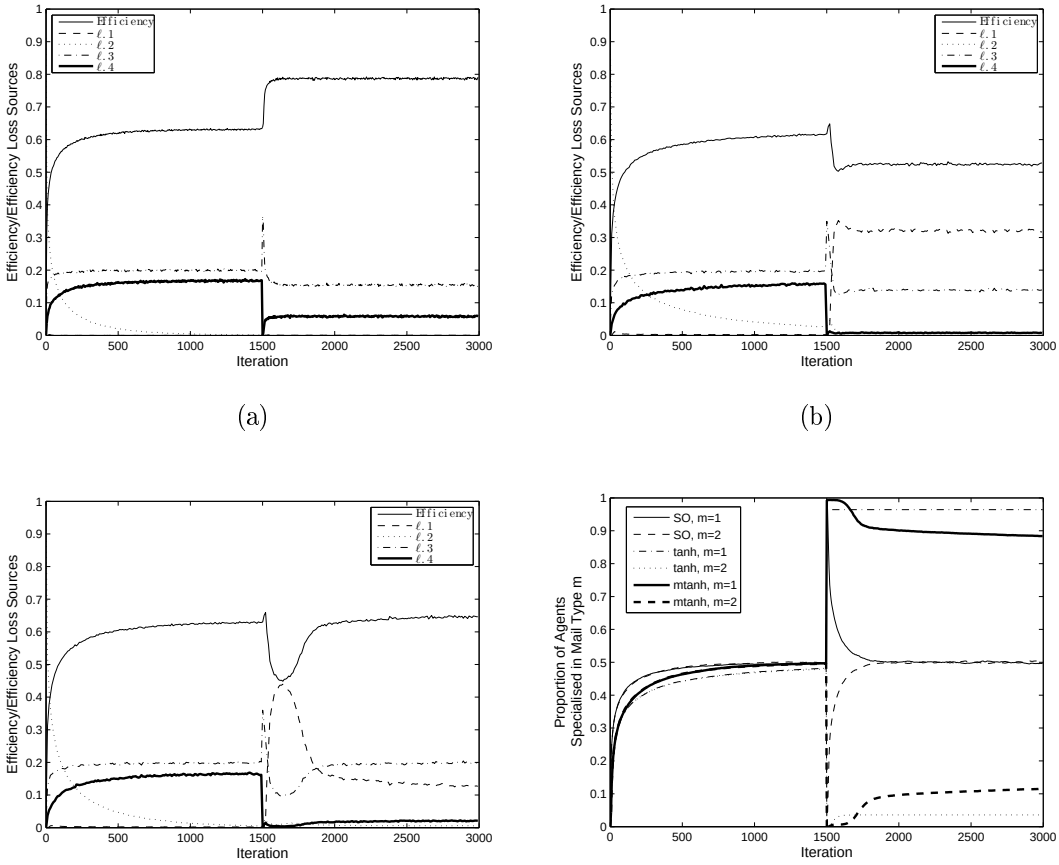


Figure 3.3: Efficiency and loss sources in a static environment with removal of specialised agents. The SO algorithm (a) almost immediately returns to the optimal split in specialisations. Due to saturation, the tanh algorithm (b) is unable to re-specialise, resulting in a dramatic increase in consecutive changeovers ($\ell.1$). Although the mtanh algorithm (c) is capable of re-specialising, it does so far less efficiently than the SO algorithm and initially reacts similarly to the tanh algorithm. (d): evolution of the fraction of specialised agents for the various algorithms.

specialise. Once enough agents are re-specialised that the average stimulus of type 2 mail drops to a level where frequent changeovers are unlikely the re-specialisation slows significantly meaning that an optimal set of specialisations will not be regained.

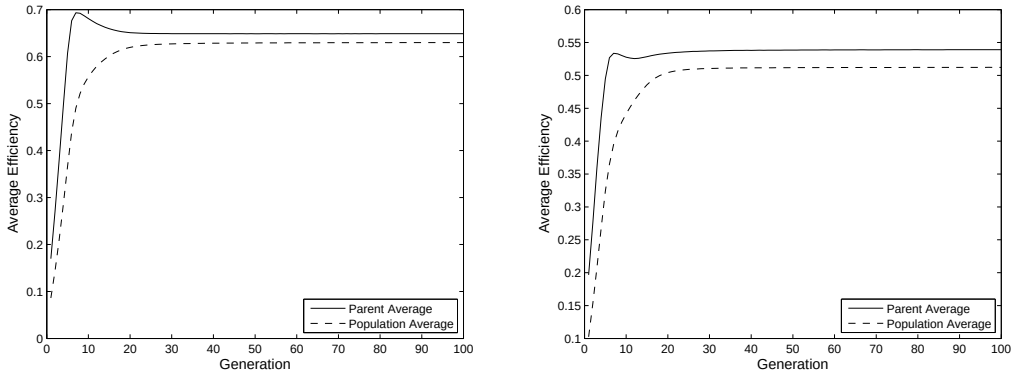
3.3.5.2 Evolutionary Optimisation

As explained in section 3.3.3, we employ an ES algorithm to obtain the optimal parameters and allow inter-update-rule competition. We use re-parametrisation to obtain better ES performance, both for parameters with fixed relationships with other ES variables ($p_1 = \theta_{max} - \theta_{min}, p_2 = \alpha/p_1, p_3 = \beta/p_1$) and for those with exponential dependence ($p_4 = \log(\lambda)$). Parameters are constrained to $\theta_{min}, p_1, \epsilon, \psi \in [0, 100]$, $p_4 \in [\log(1), \log(10)]$, and $\eta \in [-1, 1]$. It turns out that the optimised SO algorithm outperforms the other algorithms in virtually all circumstances. The only update rules that can compete with it are those that can effectively mimic its behaviour by extreme choices of parameters. As this against the spirit of the nature inspired VRT algorithm, we have constrained its parameters to $p_2, p_3 \in [0, 0.5]$.

In the ES we use a $(\mu + \ell)$-ES, with $\mu, \ell = 5000$ and initialise each element of $\boldsymbol{\sigma}$ to the size of parameter space. As mtanh can evolve into tanh for $\eta = 1$, we do not explicitly use the tanh rule in our algorithm. The ES was run for 100 generations and results are averaged over 50 runs. We investigate the average fitness both of all the agents competing in an ES generation, but also of the parent agents (i.e. those selected by the ES as parents for the next generation). We also look at the relative fractions of each update rule within the population.

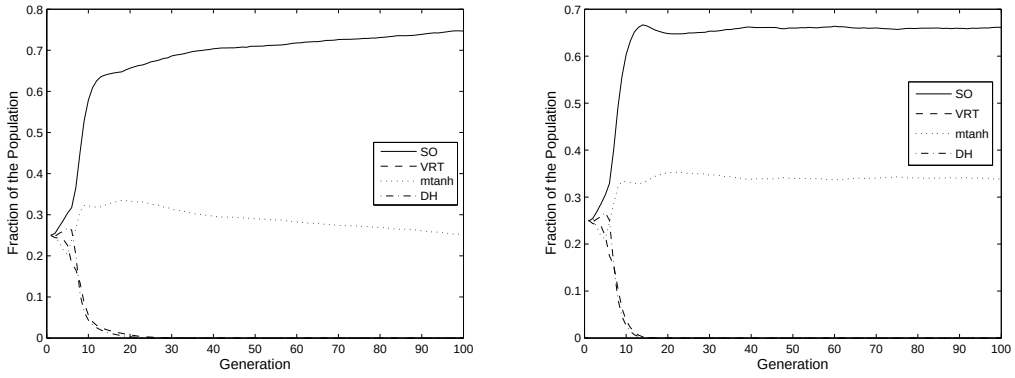
In the both environments (see figure 3.4), the ES quickly obtains a high level of efficiency. This is obtained by dropping θ_{max} to a much lower value than intuitively expected (and used in the standard setting). The remaining efficiency gain is mainly a consequence of the increasing fraction of the population with a good rule set (see figure 3.5). We observe a peak in the efficiency of parent agents during the early generations as agents first discover good rules and parameter sets. These well optimised agents have more mail available than those in later generations, as they are competing with inferior agents.

We see from figure 3.5 that the two best algorithms are the SO and the mtanh algorithms (with $\eta < 0$ and large ϵ, ψ, thus approximating SO). The SO algorithm, however, has the added advantage of not being able to move away from this be-



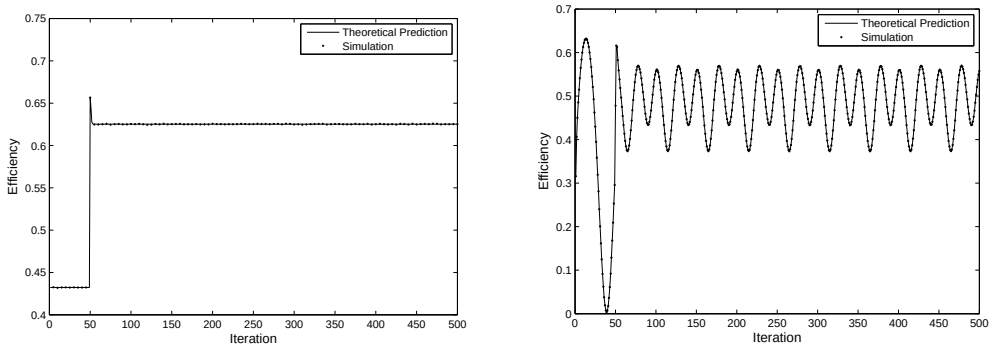
(a) (b)

Figure 3.4: Evolution of the efficiency during an ES optimisation of the population of agents in the static (a) and dynamic (b) environment. In environments both ES leads to increase the efficiency of both on average and of parent agents. After ≈ 30 generations efficiencies tend to a stable value. We also observe a peak in parent efficiency early in the run.



(a) (b)

Figure 3.5: Evolution of the population frequencies during an ES optimisation of the population of agents in the static (a) and dynamic (b) environment. In both environments, all update rules eventually tend to extinction, except the mtanh and SO rule which effectively become the same. The relative fractions are determined by the initial conditions and sensitivity to mutations.



(a) (b)

Figure 3.6: Comparison of the theoretical solution (lines) for an infinite system, with simulations (dots) in the standard static (a), and the dynamic (b) environment. Note that all agents were initialised with specialisation in mail type 1.

haviour. As such, in both environments, the SO algorithm has an initial advantage over mtanh. In the static environment the ability of mtanh to mutate away from SO style behaviour causes the fraction the mtanh agents to be slow out-competed by SO, although in the dynamic environment we see no bias towards either rule.

An analysis of the inter-run variance shows that the final proportions of SO and mtanh algorithms are only an average trends, and in some runs the fraction of mtanh agents increases from its early value, while in others it decreases. The variance is low for the first ≈ 10 generations, while VRT and DH become extinct and mtanh finds SO-like behaviour, but subsequently increases. In fact the final proportions ± 1 standard deviation are 0.747 ± 0.257 (0.661 ± 0.189) for the SO algorithm and 0.253 ± 0.257 (0.339 ± 0.189) for the mtanh algorithm in the static (dynamic) environment.

3.3.5.3 Theoretical Validation

In figure 3.6, we show the excellent agreement between the exact theory for infinite system size and simulations of a large but finite population with corresponding settings. Note that we have opted to show the efficiency, but any other quantities such as fractions of specialised agents can also be calculated and are in equally good agreement.

Table 3.1: The average efficiency of the different methods

Environment	Static	Dynamic
VRT	0.501	0.412
SO	0.586	0.467
ES	0.629	0.512
Theory	0.633	0.504

3.3.5.4 Performance

In Table 3.1, we illustrate the effect on the efficiency made by the introduction of new update rules and evolutionary optimisation in comparison to the original VRT model, which already outperforms a range of other general purpose algorithms [56; 57]. These efficiencies are given in the static and dynamic environment. Efficiencies are averaged over 500 iterations (including the initial specialisation period). ES results are given as an average over 50 runs of the performance of all the agents in the final (100^{th}) generation. Note that these figures include the damage caused by mutations and the introduction of random agents. The theoretical results are given for the optimal (integer) values of θ_{max}, which were determined by exhaustive search.

The SO algorithm already provides a large improvement over the VRT algorithm, while the ES determined rules and parameters obtain increased performance, particularly in the dynamic environment. The best results obtained by the theoretical model are very close to the final results of the ES.

3.3.6 Conclusions

In this section, we have studied a threshold based algorithm for distributed task allocation retrieval. The efficiency and flexibility have been investigated both in static and dynamic environments. and with respect to catastrophic breakdowns of agents. We have introduced new rules for mail selection and specialisation and have used a evolutionary algorithm to optimise these further. We have shown that some of the new rules have improved performance compared to existing ones. The

best ones give increased efficiency by 25.5% in a static, and 24.3% in a dynamic environment, compared to a method (VRT) which already outperformed a variety of other algorithms [56].

3.4 A Comparative Approach

While we have shown the SO algorithm to be able to outperform other threshold based rules, we do not know how it compares to other task allocation methods. Therefore, in this section we introduce another current approach to efficient task allocation, a market based algorithm. Such algorithms have been used as a control mechanism in a wide range of multi-agent systems [17]. They are characterised by their ability to assign tasks and resources using market-like mechanisms, typically in the form of an auction. While this approach does require some level of communication between agents (hence it is not applicable to all problems), if desired, this can be kept at the local level, avoiding the scalability issues associated with centralised control. We also present a novel hybrid algorithm which combines both threshold and market approaches.

As both approaches have strengths and weaknesses, we compare them in a variety of circumstances. In particular, we investigate them with regards to efficiency, flexibility (ability to react to changes in the environment), and sensitivity to load (ability to cope with differing demands). As all these approaches depend on parametrised functions, we optimise these parameters using a nature inspired particle swarm optimisation (PSO) algorithm, which allows us to determine the best rule in any given circumstance in an unbiased manner. We also wish to know how close our algorithm is to optimal efficiency, therefore we derive a theoretical upper limit on algorithm performance and include this in our comparisons when appropriate.

3.4.1 Market Based Algorithms

Markets are an area which have provided inspiration for the design of self-organising systems. **Market based (MB)** algorithms treat their elements as self-interested individuals with goals. Groups of individuals use auction-like mechanisms to decide who fulfils these goals, with bids based on both the individual's desire and ability

to complete their goal. Individual goals are designed to further the overall goals of the system, meaning that these auctions can allow agents to coordinate and produce desirable system-wide behaviour.

The market based algorithm investigated in this chapter, introduced by Campos et al. [11], approaches the problem from a different perspective than threshold based algorithms. It attempts to match each task to the best available individual, rather than allowing allowing individuals to choose tasks independently. When a group of individuals encounters set of tasks, they are all required to submit bids for the tasks, which are then assigned to the highest bidder. Hence, high bids should indicate willingness and ability to complete the task.

In comparison to threshold based algorithms, this can have both advantages and drawbacks. As individuals *must* engage in a task (if one is available, and they are capable of doing so), the maximum number of tasks is completed at each point in time when considered in isolation. However, forcing individuals to engage in tasks for which they are not suited, does incur penalties which may be costly in the long term.

The MB algorithm that we use, can be seen as a refinement of the greedy style algorithm outlined at the end of section 3.2. In the market based framework, originally developed for this problem type by Morley [49], the set of all agents at a city are offered batches of available mail in a random order. Each agent must submit a bid, determined by a *bidding function*, for the offered batch of mail, with the highest bidder being assigned the batch. In the case of equal high bids, the winner is selected at random from the highest bidding agents. This means that, if there are sufficient agents available at a city, all tasks must be assigned in a similar fashion to the greedy algorithm. The added selectivity is determined by the agents' bidding function.

We use a version of the bidding function developed by Campos et al. [11]. This function causes an agent a to submit large bids for a task with type m and waiting time w if

1. the task has a high priority (w is high)
2. the task type matches its specialisation ($\sigma_a = m$)
3. the agent can process the batch of mail soon (small backlog in queue)

It is of the form

$$B_a(w,m) = \frac{\omega_p w(1+\omega_s \delta_{m,\sigma_a})}{T_a(m)^{\omega_t}} \tag{3.17}$$

where ω_p and ω_s weight the bid in relation to the priority of the task for completion and the selectivity (desire to take mail of its specialised type) of the agent respectively. We define

$$T_a(m) = \sum_{l=1}^{L_q} q_{a,l} + \delta_{m,\sigma_a} t_p + (1-\delta_{m,\sigma_a}) t_c \tag{3.18}$$

as the time it would take agent a to clear its queue and finish processing a batch of mail of type m, which is exponentially weighted by ω_t.

As the set of agents is homogeneous and the acceptance of a bid is only determined by its magnitude relative to other bids, the priority w and weight parameter ω_p can be removed from eq. (3.17), to yield:

$$B_a(m) = \frac{1+\omega_s \delta_{m,\sigma_a}}{T_a(m)^{\omega_t}} \tag{3.19}$$

3.4.2 A Hybrid Algorithm

To maximise efficiency, we also consider a simple hybrid of the VRT and MB algorithms. In this algorithm all agents have a set of thresholds. However, the order of the action of the agents is determined in an auction identical to the MB algorithm described above and using bidding function (3.19). After each round of bidding, the winning agent is then offered the mail type it has bid for which it accepts or rejects using the ETF. In the case of rejection, it then looks at the other mail at the city in a random order, accepting or rejecting according to the ETF exactly as it would in a standard threshold based algorithm. Upon acceptance, it updates its thresholds according to the VRT update rule in eq. (3.6).

We note that this combination of VRT and market rules can behave exactly like any of the previously described algorithms. If we have $\theta_{max} = 0$ then agents will always accept the first batch of mail offered to them (i.e. the one that they have won an auction to be offered) thus making the algorithm entirely MB. Likewise if we choose $\omega_s = \omega_t = 0$ then agents will all submit identical bids of 1 to the auction, causing them to act in a random order exactly as in a pure VRT algorithm. As has already been noted, the SO algorithm is identical to the VRT algorithms with $\epsilon = \psi = \theta_{max} - \theta_{min}$.

3.4.3 Particle Swarm Optimization

For any given environment, the behaviour of the previous algorithms is governed by a set of parameters. For a fair comparison of the performance of the algorithms it is necessary to ensure that none is unfairly disadvantaged by poor parameter choices. Therefore, we optimise the parameter sets to determine the best performance of each algorithm in a given circumstance.

Unlike in the previous section, in which we were interested in competition between algorithms, we are only interested in comparisons of efficiency between our algorithms. Because of this we optimise at the population level, to avoid selective pressure on agents to improve their own performance at the cost of the rest of the population. As this is more costly in terms of function evaluations (a single run of the algorithm produces 1 result, rather than N_a results) we will use PSO to optimise our algorithms, as it typically requires fewer function evaluations than a naive GA to find a similar quality result. Since we optimise a stochastic system we need to ensure that our algorithm is robust with respect to stochasticity. Therefore we use the noise resistant PSO variant described in section 2.2.2.5

3.4.4 Theoretical Efficiency Limit

As described in section 3.3.4, it is possible to derive a complete theoretical description of a solution to our problem. However, this is dependent on a particular threshold based rule and parameter set. However, by following a similar strategy, we can derive algorithm-independent theoretical upper bounds for the efficiency of an infinite population in *ideal circumstances*, i.e. when no mail is lost due to $\boldsymbol{\ell}$.1-$\boldsymbol{\ell}$.3. This situation would occur when $t \leq L_q$ and when agents *never* reject mail such that the efficiency is only limited by $\boldsymbol{\ell}$.4.

Although this assumption does neglect an important source of complexity in the system, it does allow us to study the performance lost due to agents' random city choices. We will see in the results section that this is a driving factor in efficiency changes under the variation of certain system parameters. We will also see that in certain situations, our assumptions become realistic and that this limit does in fact match algorithm performance.

Then, both the agent profile and the mail waiting times become irrelevant and

the efficiency is a function of the profile of the following simplified city micro-states alone:

$$\mathcal{C} = \boldsymbol{b}_{N_m} = (b_1, .., b_{N_m}) \ , \tag{3.20}$$

where $b_i \in \{0, 1\}$ is the availability of mail type i at the city. The set $\mathcal{S}_\mathcal{C}$ of all possible states has cardinality $|\mathcal{S}_\mathcal{C}| = 2^{N_m}$. Defining the states of the cities as $\boldsymbol{S}(t) = \{S_c(t), c = 1, .., N_c\}$, at any time t the global state of the system is completely determined by the city profile $\boldsymbol{\chi}(t) = \{\chi_\mathcal{C}(t),\ \mathcal{C} \in \mathcal{S}_\mathcal{C}\}$, where

$$\chi_\mathcal{C}(t) \equiv \frac{1}{N_c} \sum_{c=1}^{N_c} \delta_{S_c(t),\mathcal{C}} \ . \tag{3.21}$$

Again, $\chi_\mathcal{C}(t)$ becomes a deterministic quantity in the large system limit, and we can derive the exact time evolution. In this case, we only require breaking up this time evolution into two distinct steps:

1) changes to the $\chi_\mathcal{C}$ during mail uptake.

2) changes to the $\chi_\mathcal{C}$ during mail production.

The change in city profile during mail uptake can be described by multiplication with a matrix $\mathbf{L}$, while the change in city profile during mail production can be described by multiplication with a matrix $\mathbf{P}(t)$ which is time dependent for the dynamic environment only. Combined, this gives the following exact time evolution for the city profile:

$$\boldsymbol{\chi}(t+1) = \mathbf{P}(t)\ \mathbf{L}\ \boldsymbol{\chi}(t) \ . \tag{3.22}$$

Then, the efficiency $E(t)$ (the probability that an agent takes mail at time t) is given by

$$E(t) = \sum_{k=1}^{N_m} \chi_k(t) \left(1 - P_{R_{a/c}}(k) + \frac{k - R_{a/c}}{R_{a/c}} \sum_{j=k+1}^{\infty} P_{R_{a/c}}(j) \right), \tag{3.23}$$

where $\chi_k(t) \equiv \sum_{\boldsymbol{b} \in \mathcal{S}_\mathcal{C}} \chi_{\boldsymbol{b}}(t) \delta_{|\boldsymbol{b}|,k}$ is the probability that a city has exactly k pieces of mail available, P_λ is the Poisson distribution with parameter λ, and $R_{a/c}$ is the ratio of agents to cities. The details of these derivations and the exact expressions of the matrices $\mathbf{L}$ and $\mathbf{P}(t)$ can be found in appendix A.2. A comparison between the performance of the various algorithms with this theoretical upper bound is presented in the following section.

3.4.5 Results

In this section we discuss the numerical results. Rather than looking at all threshold based algorithms again, we will only compare the MB algorithm with the best performing update rule, the SO algorithm, and the original VRT algorithm, which we will use as a baseline. First, we compare the general tendencies of the algorithms, looking at finite size effects, the behaviour over the course of a run, and the reaction to different values of N_m. Secondly, we test their ability to cope with different levels of load, in particular the reaction to the system parameters $R_{a/m}$, and t_c. Thirdly we look at the adaptability of the algorithms, testing their ability to react to continuous change in the dynamic environment, and to sudden change with the removal of specialised agents. Finally, we optimise the algorithms' parameters using PSO and compare their absolute performances in terms of overall efficiency.

As there are many parameters to cover, we have opted to investigate the influence of different factors on the efficiency systematically, by varying one parameter at a time and keeping the rest in the standard MB setting as defined in section section B.1, with $N_a = 5 \cdot 10^4$.

For the MB algorithms we take values approximately equal to the values found using a GA by Campos et al. [11], with $\omega_s = 1750$ and $\omega_t = 4$. A standard run consists of 500 iterations over which the average efficiency per agent is monitored, and the standard dynamic environment has a period $\xi = 50$.

The results presented here are intended to be representative of a larger investigation and, as such, we make the following comments. As the parameters we have chosen are by no means guaranteed to be ideal for a given setting, comparing the the algorithms quantitatively is not appropriate (until section 3.4.5.4, when we optimise the parameters). Instead we are interested in comparing the qualitative trends of the algorithms in various circumstances. Therefore, we only present the same test under multiple circumstances (e.g. high or low t_c, static or dynamic environments) if it leads to qualitatively different behaviour. The only exception is that, in almost all circumstances, high t_c causes a breakdown in the performance of the MB algorithm. We illustrate this when investigating t_c and do not restate it for our other tests. Also, in single run examples we present SO as representative of our threshold based algorithms unless VRT has qualitatively different behaviour.

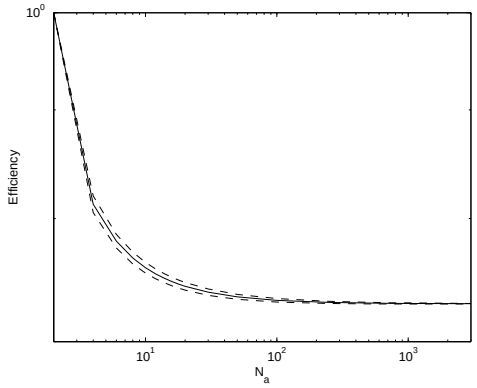


Figure 3.7: The average efficiency (solid line) of the MB algorithm as a function of the system size N_a with fixed $R_{a/m} = 1$ and $N_m = 2$, in a static environment on a log-log scale. Runs were averaged over $2 \times 10^5/N_a$ runs with error bars of $\pm$1s.d. (dashed lines).

Note that, over the course of our investigation, we observed that changes in efficiency caused by varying a system parameter were largely triggered by changes in: $\boldsymbol{\ell}.1$ in the case of the MB algorithms; $\boldsymbol{\ell}.3$ in the case of the SO algorithm; $\boldsymbol{\ell}.1$ and $\boldsymbol{\ell}.3$ in the case of the VRT algorithm. However, $\boldsymbol{\ell}.3$ for the VRT algorithm exhibits qualitatively similar behaviour to $\boldsymbol{\ell}.3$ for the SO algorithm. Therefore, for the sake of clarity, when presenting all three algorithms on the same graph we will only include $\boldsymbol{\ell}.1$ for market and the VRT algorithm and $\boldsymbol{\ell}.3$ for the SO algorithm unless there is good reason to do otherwise.

3.4.5.1 General tendencies

In order to ensure that our results on FSE effects in threshold based algorithms also hold for the MB algorithm, we vary N_a, while keeping N_m and $R_{a/c}$ fixed. The results, shown in figure 3.7, are qualitatively similar to our threshold based results (figure 3.1) so we can conclude that our system size of $N_a \simeq 10^4$ is still sufficient.

In the static environment (see figure 3.8), the market based algorithm maintains a constant high efficiency throughout, with all loss caused by $\boldsymbol{\ell}.4$ which, as shown in section 3.4.4, is unavoidable. This contrasts with the SO algorithm (figure 3.2 a), which is also affected by $\boldsymbol{\ell}.3$.

Figure 3.9 shows the efficiency as a function of N_m. For the threshold based algorithms, the efficiency initially increases with N_m due to a more uniform distribution of agents over cities, decreasing the likelihood of $\boldsymbol{\ell}.4$. For the SO algorithm

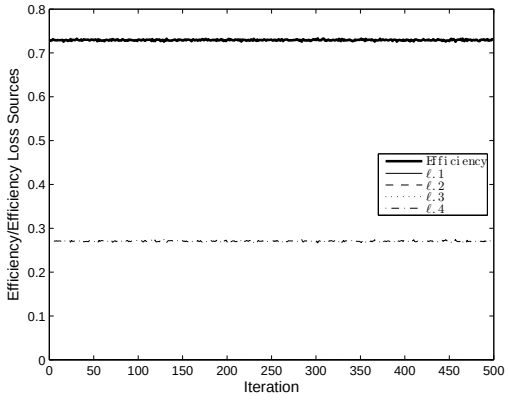
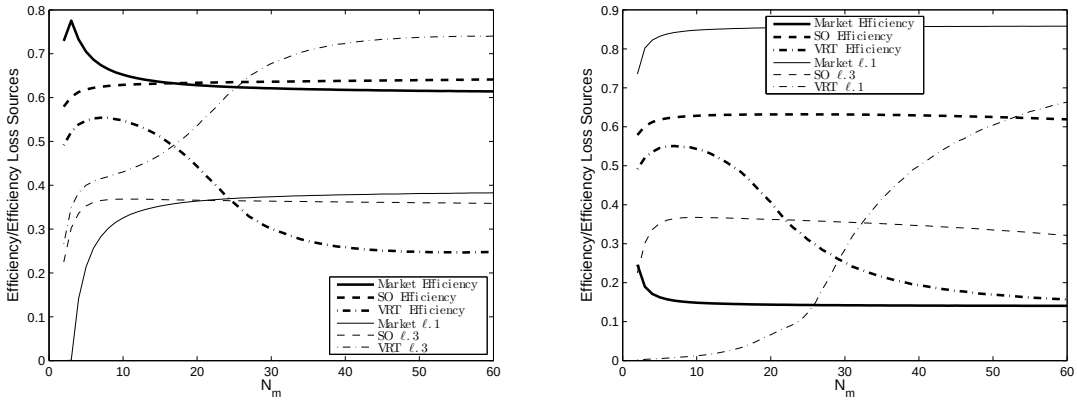


Figure 3.8: Evolution of the efficiency and loss sources during a single run in the static environment using the MB algorithm. The behaviour of the MB algorithm exhibits stable efficiency throughout, and all loss source except $\boldsymbol{\ell}.4$ are negligible.



(a) (b)

Figure 3.9: Efficiency and loss sources as function of N_m in the static environment for $t_c = 2$ (a) and $t_c = 10$ (b). In both situations the threshold based algorithms show an initial increase in efficiency, while the MB algorithm shows a sharp increase followed by a decrease, as $\boldsymbol{\ell}.1$ increases, for $t_c = 2$ and a sharp decrease for $t_c = 10$. Both the MB the SO algorithm quickly tend to a relatively stable efficiency, although market exhibits a small downward trend driven by increasing $\boldsymbol{\ell}.3$ and SO a small upward trend driven by decreasing $\boldsymbol{\ell}.3$. After the initial increase the VRT algorithm decreases in efficiency, before tending towards a stable value. For $t_c = 2$ this is caused by $\boldsymbol{\ell}.3$ while for $t_c = 10$ this is caused by an increase in $\boldsymbol{\ell}.1$ (note the change in between (a) and (b)).

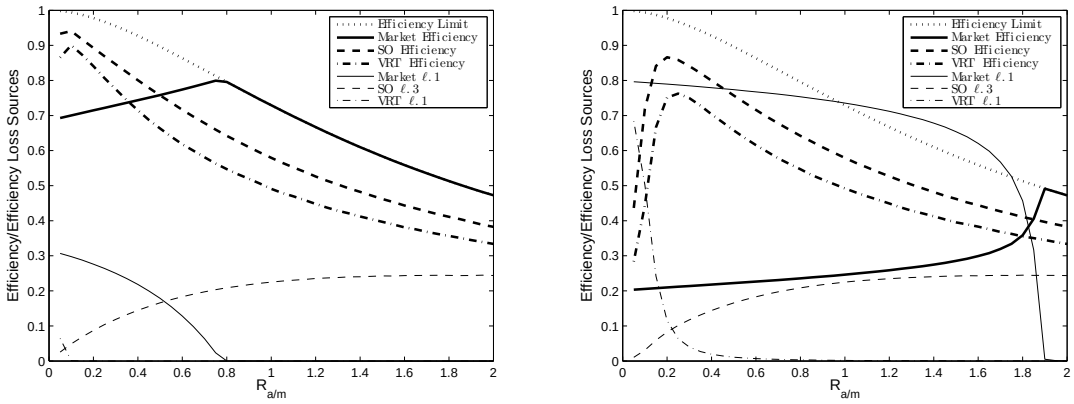
the efficiency then levels off, while for the VRT algorithm it decreases as N_m further increases.

For $t_c = 2$ this is caused by an increase in $\boldsymbol{\ell}.3$. At low levels of N_m this matches an increase in $\boldsymbol{\ell}.3$ for the SO algorithm and can be explained by a higher probability of encountering other mail types (i.e. a trade-off with $\boldsymbol{\ell}.4$). The additional increase on top of SO's $\boldsymbol{\ell}.3$ is explained by VRT's lower ability to re-specialise. VRT agents rely on repeated exposure to high stimulus batches of mail of a particular type to force them to change their thresholds to match their specialisation. When N_m is high (and, therefore N_c is low for fixed $R_{a/m}$) there is low chance that a particular agent will repeatedly see a batch of the same under-served mail type. This leads to low pressure to develop an optimal split in specialisations (N_a/N_m agents specialised in each mail type) and an increased chance for similarly specialised agents to visit the same city (high $\boldsymbol{\ell}.3$).

For $t_c = 10$, VRT's loss of efficiency is due to an increase in $\boldsymbol{\ell}.1$. For high N_m, agents must examine more mail before they find their type leading to increased chances of changeovers. As VRT agents are slow to change their thresholds to match their specialisation, this is likely to lead to repeated changeovers and thus, when t_c is high, increased $\boldsymbol{\ell}.1$.

The MB algorithm experiences an initial increase in efficiency for $t_c = 2$, for the same reason as the threshold based algorithms: a decreased chance of $\boldsymbol{\ell}.4$ due to the increasingly uniform numbers of agents at cities. The following increase in $\boldsymbol{\ell}.1$, and levelling off of efficiency, is caused by agents being forced to bid upon more mail before they are offered their specialised type. For $t_c = 10$, the MB algorithm shows an initial increase in $\boldsymbol{\ell}.1$, merely due to the fact that an increasing fraction of available mail will not be of the specialised type. The efficiency loss then levels off because $\boldsymbol{\ell}.1$ approaches 0.9, the total fraction of $\boldsymbol{\ell}.1$ that would occur if agents always switched mail types with $t_c = 10$. This limit is never reached as market agents don't always have the opportunity to take mail.

While it is important to note which parameter values cause a breakdown in the algorithms (e.g. high t_c in the MB algorithm), behavioural comparisons are more relevant under *normal* conditions. As high N_m causes a breakdown in the VRT algorithm and, for appropriate t_c, leads to relatively similar behaviour in the SO and the MB algorithms, we fix $N_m = 2$ in the remainder of our investigation.



(a) (b)

Figure 3.10: Efficiency and loss sources as function of $R_{a/m}$ for $N_m = 2$ in the static environment for $t_c = 2$ (a) and $t_c = 10$ (b). For the threshold based algorithms, and with the exception of very low values of $R_{a/m}$, where $\boldsymbol{\ell}.1$ dominates, the efficiency follows the same trend as the theoretical upper bound. As expected loss $\boldsymbol{\ell}.1$ becomes negligible and $\boldsymbol{\ell}.3$ increases as $R_{a/m}$ increases. The deviation from the theoretical limit is less pronounced for $t_c = 2$. For the the MB algorithm efficiency increases with $R_{a/m}$, coupled with decreasing $\boldsymbol{\ell}.1$, until it reaches the theoretical limit. It then immediately switches to a downward trend taking the same value as the limit.

3.4.5.2 Load

The system's effective *load* is determined by how close it is to a breakdown due to $\boldsymbol{\ell}.1$. This is a self-reinforcing concept, as high $\boldsymbol{\ell}.1$ increases load on the remaining agents, increasing $\boldsymbol{\ell}.1$ further. The parameter $R_{a/m}$ is closely linked to the load as it is effectively a measure for the demand on agents.

For low $R_{a/m}$ there are many more tasks than agents and demand is high such that the breakdown of a few agents will cause a proportionally larger increase in load than for large $R_{a/m}$. The parameter t_c also determines the sensitivity of the system to load. For high t_c overloaded agents suffer $\boldsymbol{\ell}.1$ more often, and remain inactive for a longer period of time than for low t_c. Therefore, we investigate the effects of both parameters.

In figure 3.10, we compare the upper bound with the actual efficiency and the loss sources of the algorithms (for $N_m = 2$), as a function of $R_{a/m}$. For threshold based algorithms the low efficiency at low $R_{a/m}$ is due to high average waiting times which become close enough to θ_{max} to overwhelm agents' selectivity and force multiple changeovers and high levels of $\boldsymbol{\ell}.1$. Note that this is merely a function of our arbitrarily chosen θ_{max}, progressively increasing this value as $R_{a/m}$ decreases

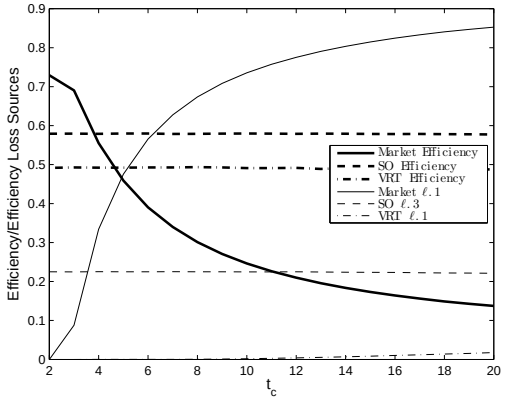


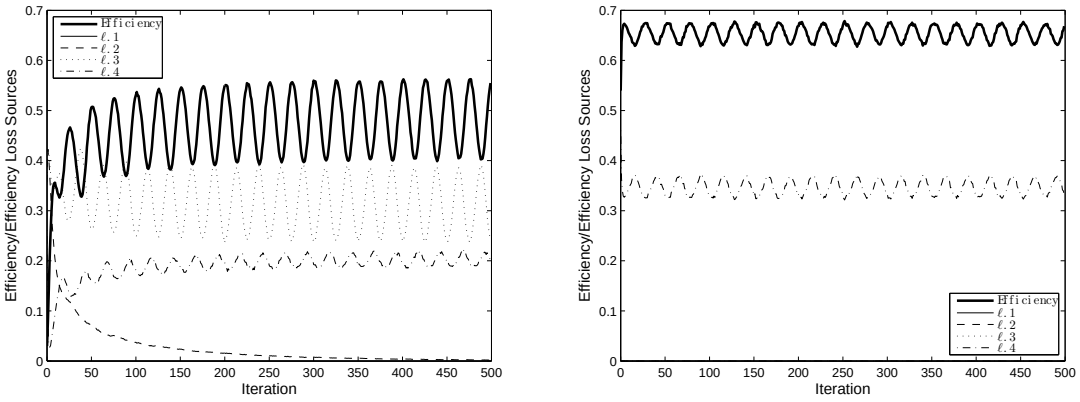
Figure 3.11: Efficiency and loss sources as function of the changeover time t_c in the static environment. Increasing t_c causes $\boldsymbol{\ell}.1$ to increase for MB algorithms, with a corresponding drop in efficiency. The value of t_c has little effect on the threshold based algorithms, its only noticeable result being a small increase in $\boldsymbol{\ell}.1$ for the VRT algorithm while other values remain approximately static.

would allow us to overcome this lack of selectivity. At high values of $R_{a/m}$ it is clear that θ_{max} is too high as we know that a smaller population could serve the demand, such that a drop in $\boldsymbol{\ell}.1$ would be acceptable in order to decrease $\boldsymbol{\ell}.2$ and $\boldsymbol{\ell}.3$.

For the MB algorithm a high demand upon agents (i.e. low $R_{a/m}$) leads to $\boldsymbol{\ell}.1$. Therefore, as $R_{a/m}$ increases $\boldsymbol{\ell}.1$ decreases, leading to increasing efficiency until $\boldsymbol{\ell}.1$ reaches a negligible value. At this point $\boldsymbol{\ell}.1$-$\boldsymbol{\ell}.3$ are all negligible satisfying the conditions used to derive the theoretical limit on efficiency. Thus, the algorithm's efficiency increases until it hits this limit after which it saturates the theoretical limit as $\boldsymbol{\ell}.4$ entirely determines its behaviour.

Note that while the increased changeover penalty from figure 3.10 (a) to (b) makes only a small difference in the demand that threshold based algorithms can cope with, it renders the MB algorithm unproductive up to a relatively high value of $R_{a/m}$. After this value $\boldsymbol{\ell}.1$ drops sharply due to its self reinforcing nature.

We can see from figure 3.11 that the level of t_c has very little effect on the threshold based algorithms. These algorithms tend to reject mail which is of a different type to their effective specialisation, and the penalty caused by high t_c is incurred infrequently, leaving the agents with time to clear their queue backlog. The performance of the MB algorithm, however, is highly determined by t_c. As the algorithm is incapable of rejecting mail offered to it by a city, it incurs a far higher proportion of changeovers than the threshold based algorithms. This leads



(a) (b)

Figure 3.12: Evolution of the efficiency and loss sources during a single run in the dynamic environment using the SO (a) and MB (b) algorithms. The values of the loss sources and the efficiency fluctuate around their *average* values, which are qualitatively similar to those in the static environment.

to a huge loss in performance as $\boldsymbol{\ell}.1$, the proportion of agents inactive due to a full queue, increases with t_c.

To conclude, we observe that the threshold based algorithms can cope reasonably well with a much greater range of loads. Although the MB algorithm manages to saturate the bound in conditions of low load, it completely breaks down under high load.

3.4.5.3 Adaptability

To investigate the ability of the algorithms to cope with changing environments, we also test them in the continuously changing dynamic environment as introduced in eq. (3.1), and with respect to a sudden change such as the removal of all specialised agents in one mail type.

In the dynamic environment (see figures 3.12,3.2 (c)), we observe variations in efficiency over the course of a wavelength. The positions of the minima and maxima are surprising at first sight, as the total probability for mail production remains static and the points of maximum efficiency occur where a non-uniform distribution of mail is expected. However, at the end of an iteration in which mail type m is predominantly produced it is also more likely for this mail type to be left over. This in turn lowers the probability of this mail type being produced in the next iteration compared to when $\pi_1(t) \approx \pi_2(t)$. Hence, while the a priori total probability of mail

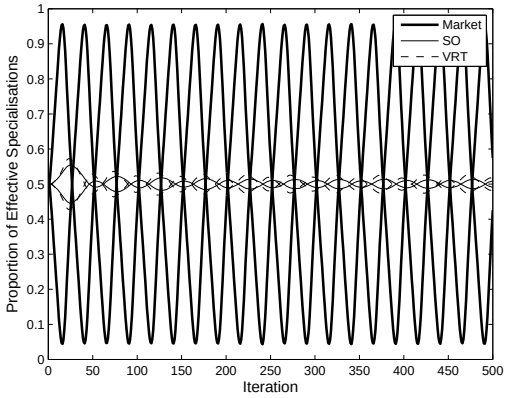


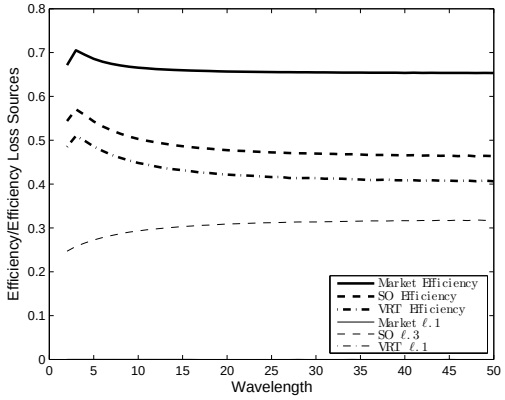
Figure 3.13: Specialisation behaviour in a dynamic environment using the standard MB settings. All algorithms fluctuate around the average mail production probability, 0.5. The threshold based algorithms do not fluctuate far from this average value while the MB algorithm does.

production probability is static, the *effective* total probability of mail production is maximal when $\pi_1(t) - \pi_2(t) \to 0$.

If we consider the average values rather than the details of the fluctuations, the behaviour of all algorithms is qualitatively very similar to that in the static environment. The relative values of the efficiencies and loss sources are approximately preserved, as is their evolution over the course of a run, and the MB algorithm reacts in the same manner to changes in t_c.

Figure 3.13 shows that the market and threshold based algorithms have vastly differing effective specialisation behaviour. While both threshold based algorithms maintain a roughly even split in specialisations, fluctuating somewhat with the state of the environment, the MB algorithms tends to fully re-specialise to get as close to the optimal specialisation as possible. Note that when comparing MB and threshold based algorithms we investigate the effective specialisation, rather than the extremisation of thresholds which have no meaning when applied to MB algorithms.

Figure 3.14 shows the average efficiency and loss sources as a function of the wave length for the dynamic environment. The peak in average efficiency at relatively short wavelengths has the same origin as the peaks of instantaneous efficiency seen inside a single run in the dynamic environment: persistence of mail. At short wavelengths the state of the environment changes so quickly that left over mail from the previous iteration is less likely to be of the type that is predominantly currently produced. Hence, the *effective* mail production is increased. The relatively low



(a)

Figure 3.14: Efficiency and loss sources as function of the wavelength ξ in the dynamic environment. After efficiency initially increases, it then gently decreases to the long wave length value.

efficiency at the initial value $\xi = 2$, is caused by the discrete nature of the iterations. The sine wave part of (3.1) becomes $\sin((t-m)\pi) = 0$ for $N_m = 2$, such that the mail production probability probability is effectively constant with $\pi_m = 0.5$. As in the previous results, the MB algorithm is compromised by high $\boldsymbol{\ell}.1$ in the standard setting, but performs well with $t_c = 2$.

To investigate the behaviour of the algorithms under abrupt changes, we let the system equilibrate for 1500 iterations in the standard static environment to allow agents to specialise fully. Then we remove that half of the population that is effectively specialised in e.g. mail type 2, and equilibrate the remaining system (with halved $R_{a/m}$) for a further 1500 iterations. The results, shown in figure 3.15, show that all algorithms are capable of reacting to this abrupt change.

Both threshold based algorithms show a short spike in $\boldsymbol{\ell}.3$ immediately following the loss of specialised agents as most of the mail remaining at cities will initially be of opposite type to their effective specialisation. Agents then re-specialise, causing a brief increase in $\boldsymbol{\ell}.1$ and decrease in efficiency for the VRT algorithm, sending their efficiencies to a new, higher, value appropriate to the decreased $R_{a/m}$. While the SO algorithm quickly re-specialises to an optimal split, the VRT algorithm stops short. This is because a VRT agent which has just re-specialised in mail type m will not necessarily take mail of type m with high probability as the new threshold, potentially $\theta_{max} - \epsilon$, may be high. In fact a mail type may never become fully re-specialised in unless its average waiting time becomes high enough to overwhelm

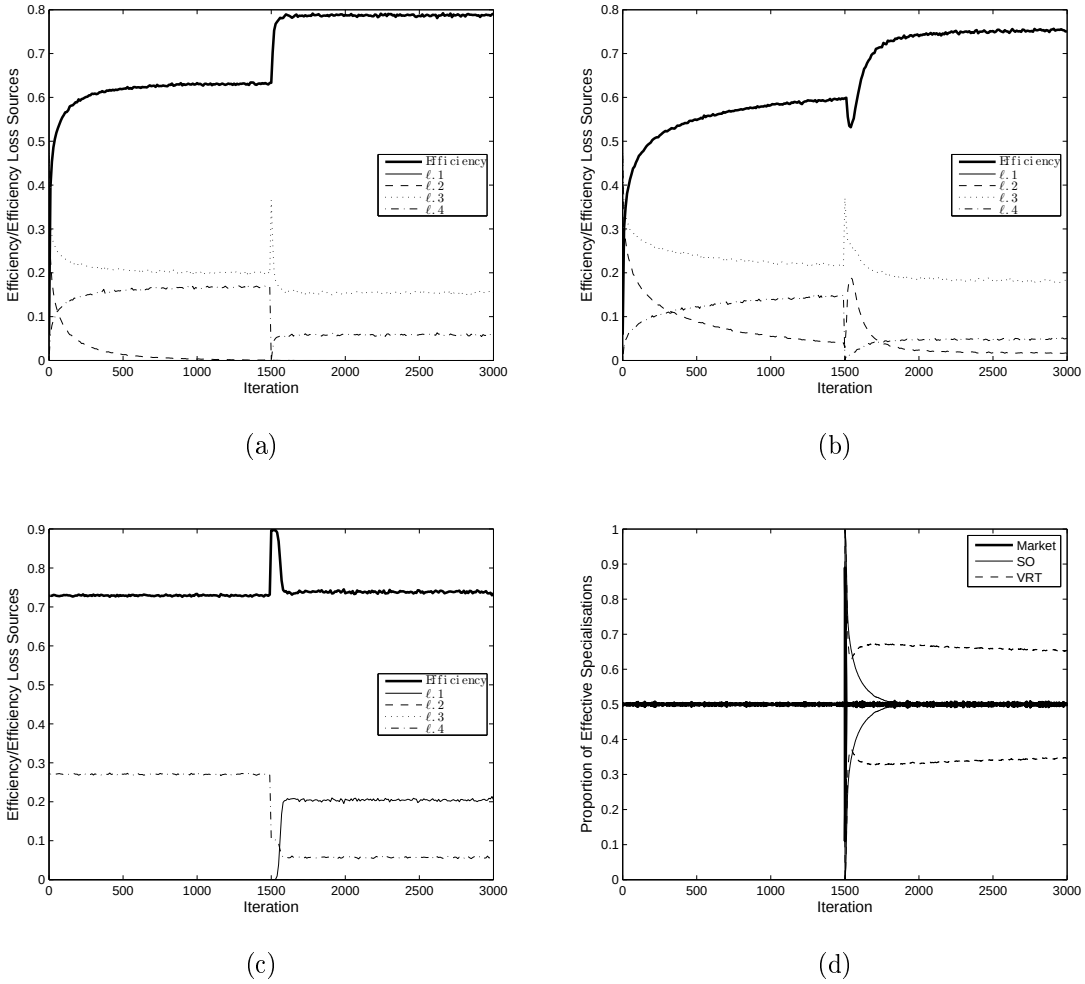


Figure 3.15: Evolution of the efficiency and loss sources during a single run with removal of specialised agents using the SO algorithm (a), the VRT algorithm (b), the market based algorithm (c), and the specialisation behaviour of these algorithms (d). After the removal of the specialised agents, both threshold based algorithms increase in efficiency following a short period with increased $\ell.3$. For VRT we see an initial drop in efficiency, corresponding to an increased level of $\ell.2$. The MB algorithm shows a long term small efficiency increase following a short term large increase corresponding to a sharp drop in $\ell.3$ and a similar increase in $\ell.1$. In (d) The MB algorithm almost immediately returns to the optimal split in specialisations, and the SO algorithm does so quickly. The VRT algorithm, however, starts towards the optimal re-specialisation, but stops short of it with a disparity remaining. Note that the after stopping re-specialising, the VRT algorithm actually moves *away* from optimal specialisation.

the selectivity of the threshold function Θ. Once enough agents have re-specialised in mail type 2, the average waiting time decreases to a manageable level, halting further progress towards optimal re-specialisation.

The MB algorithm shows an increase in efficiency as it re-specialises to suit the new conditions almost instantaneously, and the increase in the amount of available mail causes a drop in $\ell.4$. However, the high load on the system forces changeovers which in turn leads to $\ell.1$. This increase does not take effect immediately as it takes time for agents to build a backlog in their queue.

3.4.5.4 Performance

Although a full investigation of the effect of all parameters of the algorithms is beyond the scope of this chapter, we do wish to be able to compare the performance of our algorithms. Therefore, we employ a robust PSO algorithm to find optimised parameters with good performance. We cannot guarantee the parameters to be truly optimal, but they do allow each algorithm to perform well, unhampered by initial parameter choices.

As an exhaustive investigation of the optimised performance of these algorithms under all possible combinations of system parameters (or even under individual variation of each system parameter) would be impractical, we choose to test the algorithms' performances in four representative settings: the static and dynamic settings with either $t_c = 2$ or $t_c = 10$. The static and dynamic settings are representative of an algorithms' performance when stability, respectively adaptability is required. Similarly, $t_c = 2,\ 10$ are representative for low and high load situations respectively. Under low load immediate task completion is more important than the short-term performance of individual agents, while high load requires selectivity to avoid a fatal cascade of agent failures.

We optimise the performance of three different algorithms: the SO and market based algorithms (as they outperform VRT in all circumstances), and the hybrid VRT algorithm. It allows us to test whether the hybrid algorithm can outperform the individual algorithms on which it is based. Given its ability to evolve into any of the other algorithms (including the VRT algorithm), it gives us a method to determine the optimal single algorithm. Details of the optimised parameters are provided in B. Note that PSO provided no improvement the MB algorithm over the

Table 3.2: Final efficiencies (as proportions of the theoretical upper limit) with $t_c = 2$

Environment	Static	Dynamic
Original VRT	0.492156 (0.674807)	0.406784 (0.622904)
Original SO	0.579163 (0.794104)	0.463681 (0.710030)
Original Market	0.729365 (1.00004)	0.653368 (1.00050)
Optimised SO	0.704393 (0.965810)	0.653255 (1.00032)
Optimised Market	0.729369 (1.00005)	0.653368 (1.00050)
Optimised VRT Hybrid	0.72926 (0.999905)	0.653203 (1.000243)

standard settings, due to its relative insensitivity to parameter choice [40].

Table 3.2 illustrates the efficiency of our various algorithms for $t_c = 2$ in comparison to the original VRT algorithm, which already outperforms a range of other general purpose algorithms [56; 57]. These efficiencies are given in absolute numbers and as a fraction of the theoretical upper bound (in brackets). Note that fractions of the theoretical limit > 1 are possible as our system, though large, is still stochastic.

We can see that while the introduction of the original SO algorithm provides an increase in performance, the MB algorithm easily outperforms both, reaching the theoretical limit on efficiency. Optimising the SO algorithm allows it to close this performance gap, particularly in the dynamic environment and the VRT hybrid displays similar performance by finding parameters which make it entirely MB.

Table 3.3 gives the efficiency of our algorithms for $t_c = 10$. In this situation with high load, the efficiency of the MB approach falls far below that of the threshold based algorithms due to its inherent inability to reject mail. The performance gained by choosing SO over VRT is large, and we get a comparable performance boost by optimising the parameters. The hybrid algorithm slightly outperforms SO by essentially using a SO based threshold mail selection/rejection method, while using a MB approach to increase its chance of seeing its desired mail type if possible.

Table 3.3: Final efficiencies (as proportions of the theoretical upper limit) with $t_c = 10$

Environment	Static	Dynamic
Original VRT	0.491360 (0.673715)	0.405367 (0.620735)
Original SO	0.579069 (0.793975)	0.463695 (0.710052)
Original Market	0.246390 (0.337831)	0.250129 (0.383020)
Optimised SO	0.630432 (0.864400)	0.513024 (0.785589)
Optimised Market	0.246399 (0.337843)	0.249985 (0.382800)
Optimised VRT Hybrid	0.645127 (0.884549)	0.536984 (0.822278)

3.4.6 Conclusions

In this section we have studied both nature inspired and market based algorithms for distributed task allocation applied to a problem of mail processing. We have investigated the algorithms' ability to cope with load and their adaptability. In particular, we found that nature inspired, threshold based algorithms have a far higher tolerance of load than the market based algorithms. Market based algorithms were found to be quicker to adapt to system changes than the threshold based algorithms, although this only translated into a small performance difference compared to the optimised SO algorithm.

We have identified the various loss sources, and have demonstrated that the random choice of cities to visit by the agents forms the main limitation on the maximal attainable efficiency. We have derived this limit theoretically. We also investigated the absolute absolute performance of the algorithms in relation to this limit and, to that end, introduced a new hybrid approach and used a particle swarm optimisation algorithm to find good parameter sets for our algorithms. These algorithms, with optimised parameters, give us increased efficiency in a low load setting of 48.2% in a static, and 61.3% in a dynamic environment, compared to a method (VRT) which already outperformed a variety of other algorithms [56; 57]. In a high load setting these figures are 31.3% in a static, and 32.4% in a dynamic environment.

4 Task Allocation with Memory

CONTENTS

4.1 Motivation

While both standard methods (threshold and marked based algorithms) presented in section 3 can provide a good solutions to the mail processing problem at a given city, they are always limited by the likelihood of poor city choices. An agent choosing a city at random is no more likely to visit a city which has no other agents visiting it than it is to visit a city which has already had all its mail taken, thus leaving the first city unserved and the agent without mail. This puts an upper limit on the efficiency, shown in equation (3.23).

For fixed resources (number of agents) and environment (number of cities and mail types) the only way to improve on this limit is to change the profile of agents visiting cities, and hence the way in which agents choose cities. When designing an efficient method for agents to visit cities, it is useful to establish some conditions on how agents should be allocated to cities in ideal circumstances:

($\boldsymbol{c}$.1) No city should be visited by fewer agents than it has mail available.

($\boldsymbol{c}$.2) For each mail type m that a city has available, exactly one of its visiting agents should have specialisation $\sigma_a = m$.

Globally, $\boldsymbol{c}$.1 minimises $\boldsymbol{\ell}$.4 while $\boldsymbol{c}$.2 reduces the trade-off between $\boldsymbol{\ell}$.1 & $\boldsymbol{\ell}$.3. However, while a lack of appropriately specialised agents can be rectified by the threshold model, $\boldsymbol{c}$.1 is impossible to fulfil if $R_{a/m} < 1$, where $R_{a/m} = \frac{N_a}{N_c \times N_m}$ is the ratio of agents to mail. Hence, we must add a third condition to ensure that no cities go unserved for long periods of time.

($\boldsymbol{c}$.3) If $\boldsymbol{c}$.1 & $\boldsymbol{c}$.2 cannot be fulfilled over a single time step, they should be fulfilled uniformly at all cities over a longer period.

In an attempt to fulfil these conditions while maintaining the decentralised nature of the algorithm (no knowledge of the state of cities before agents visit them), and the relative simplicity of its component agents, we propose a *Stimulus Based (SB)* system of agent memory in which taking mail from a city increases an agent's chance of revisiting the city in the future.

Both market and threshold based methods provide a framework in which good global behaviour is emergent from local interactions. We can see from compara-

tive studies [11; 40], and from the work in the previous chapter, that both provide comparable efficiency in problems similar to mail processing. However, market based algorithms perform better when flexibility in mail choice is required, whereas threshold based algorithms minimise changeovers, with agents gaining strong, stable specialisations. As stable specialisations will be an important requirement of our algorithm, we have opted to base our method of task selection on the threshold model.

4.2 Memory

Each agent a is assigned a memory $\vec{\mathcal{M}}_a$, consisting of a set of μ_a paired variables $\vec{\mathcal{M}}_a \equiv \{(\mathcal{C}_{a,\ell}, \mathcal{W}_{a,\ell}),\ \ell = 1, .., \mu_a\}$, where $\mathcal{C}_{a,\ell}$ is a city which the agent has taken mail from in the past, and where $\mathcal{W}_{a,\ell}$ is a weight assigned to this memory.

An agent a either bases its selection of city on its memory $\vec{\mathcal{M}}_a$ with probability ρ_a, or chooses one randomly with probability $1-\rho_a$. The parameter $\rho_a \in [0,1]$ allows us to move continuously from a memoryless scenario ($\rho_a = 0$), to one completely dominated by the memory ($\rho_a = 1$). The total probability that agent a visits city c is given by

$$V(c|\vec{\mathcal{M}}_a) = \rho_a\ M(c|\vec{\mathcal{M}}_a) + (1-\rho_a)\frac{1}{N_c} \tag{4.1}$$

where $M(c|\vec{\mathcal{M}}_a)$ is the probability that city c is visited given that memory $\vec{\mathcal{M}}_a$ is used. Initially, the probability for choosing a specific city from the memory was taken to be its normalised weight $\frac{\mathcal{W}_{a,\ell}}{\sum_{j=1}^{\mu_a}\mathcal{W}_{a,j}}$. However, we encountered the problem that self-reinforcement tends to lead to one weight becoming so much larger than all the others that it completely dominates, thus making stable multiple city-specialisation virtually impossible. In general, this may lead to some cities being well served while others are neglected in violation of condition $\boldsymbol{c}.3$ (for similar reasons values of $\rho_a < 1$ are needed). This effect can be avoided by the introduction of a maximum usable weight $\mathcal{L}_a$, and by replacing the $\mathcal{W}_{a,\ell}$ with $\min(\mathcal{W}_{a,\ell}, \mathcal{L}_a)$, such that

$$M(c|\vec{\mathcal{M}}_a) = \sum_{\ell=1}^{\mu_a} \frac{\min(\mathcal{W}_{a,\ell}, \mathcal{L}_a)}{\sum_{j=1}^{\mu_a}\min(\mathcal{W}_{a,j}, \mathcal{L}_a)}\ \delta_{c,\mathcal{C}_{a,\ell}}\ , \tag{4.2}$$

when $\sum_{j=1}^{\mu_a}\mathcal{W}_{a,j} > 0$, and $M(c|\vec{\mathcal{M}}_a) = \frac{1}{N_c}$ otherwise. This allows for uniform probabilities in a small subset of cities (needed for $\boldsymbol{c}.3$), while allowing the weights

themselves to become large (giving the agents' city choices stability). Note that although this stability is an advantage in the current scenario, it would leave the agent vulnerable if a breakdown were to occur at a city for which it has built up a large weight.

In a similar spirit to the threshold model, we propose a memory weight update that is stimulus based. In SB memory, cities do not occur more than once in an agent's memory, such that an agent can remember up to μ_a cities. Each city in the agent's memory is assigned an individual weight which increases when an agent takes mail from that city proportionally to the stimulus (waiting time) of the taken mail, and decreases when it does not. We loosely define a city to be *well-served* if it has a set of agents which return to it repeatedly and ensure that all types of mail are taken from it with regularity. Specialisation of any new agent in a city which is already well-served adds nothing to $\boldsymbol{c}.1$, $\boldsymbol{c}.2$, and should be avoided. As agents have no direct knowledge of other agents' memories and specialisations, they must infer it from the only available information at a city, namely the waiting times. Mail at well-served cities will tend to have low waiting times compared to poorly served cities, such that it makes sense to make the increase in weights proportional to the waiting time of taken mail. Upon taking mail with waiting time w from city c the agent's memory is updated as follows:

1. If $\mathcal{C}_{a,\ell} = c$ (city c is already in the agent's memory), then its weight is increased by w:

$$(\mathcal{C}_{a,\ell}, \mathcal{W}_{a,\ell}) \to (\mathcal{C}_{a,\ell}, \mathcal{W}_{a,\ell} + w) \tag{4.3}$$

2. Otherwise if w is at least as big as the least weight $\mathcal{W}_{a,\ell}$ then city c replaces this lowest weighted element.

$$(\mathcal{C}_{a,\ell}, \mathcal{W}_{a,\ell}) \to (c, w) \tag{4.4}$$

   Note that in case of multiple equal minimum weights, only the city which was last visited the longest time ago is replaced.

3. All unmodified, non-zero weights decay.

$$(\mathcal{C}_{a,\ell}, \mathcal{W}_{a,\ell}) \to (\mathcal{C}_{a,\ell}, \mathcal{W}_{a,\ell} - 1) \tag{4.5}$$

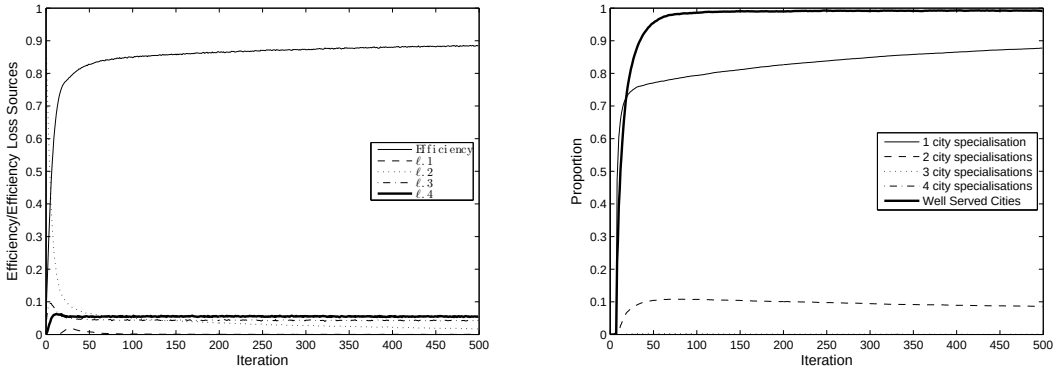
Note that if no mail is taken, all non-zero weights decay. We define an agent which has a weight of at least $\mathcal{L}_a$ in a city is *city-specialised* in that city. Note that an agent a can in principle be city-specialised in up to μ_a cities. Now, we can formalise the definition of a well-served city as a city which has an agent a specialised in it with mail specialisation $\sigma_a = m$ for every mail type m, such that $\boldsymbol{c}.1$, $\boldsymbol{c}.2$ are fulfilled locally.

If it persists, a well-served city can be seen as an example of emergent cooperation between agents. Each agent minimises the waiting time of its given mail type which decreases the chance of changeovers for other city-specialised agents. In return its own chance of a changeover is decreased by the low waiting times of all other mail types. This situation resolves the conflict between $\boldsymbol{\ell}.1$ and $\boldsymbol{\ell}.3$ and could, in principle, lead to perfect efficiency. Note that the persistence of a well-served city relies on both the stability of the serving agents' memory and their specialisations. This justifies the use of the threshold model over a market based approach to task allocation.

Assuming that N_m, $R_{a/m}$ and μ_a are fixed and finite, both the memory requirements to implement this algorithm, and the number of operations per time step, scale *linearly* with the system size N_a. An agent's behaviour (including memory) is only affected by the stimulus detected at cities and this is independent of N_a: hence, each agent performs O(1) operations. Cities must perform N_m operations to increase their waiting times and in the worst case (all agents visiting a single city) must perform $N_a - 1$ operations to randomly order the agents.

4.3 Results

While the full optimisation of the model's parameters for particular circumstances is beyond the scope of this chapter, we do study the influence of some key parameters on the system's qualitative behaviour. Parameters that are not explicitly varied, are set to those given in the standard setting (see Appendix B.1). Hence, we take $N_m = 2$ which is the most interesting case, as the distribution of agents to cities becomes more uniform with increasing N_m such that the upper limit on the efficiency (3.23) tends to 1. Furthermore, we take $R_{a/m} = 1$, as this is both the minimum ratio at which all cities could in principle be served perfectly, and the maximum



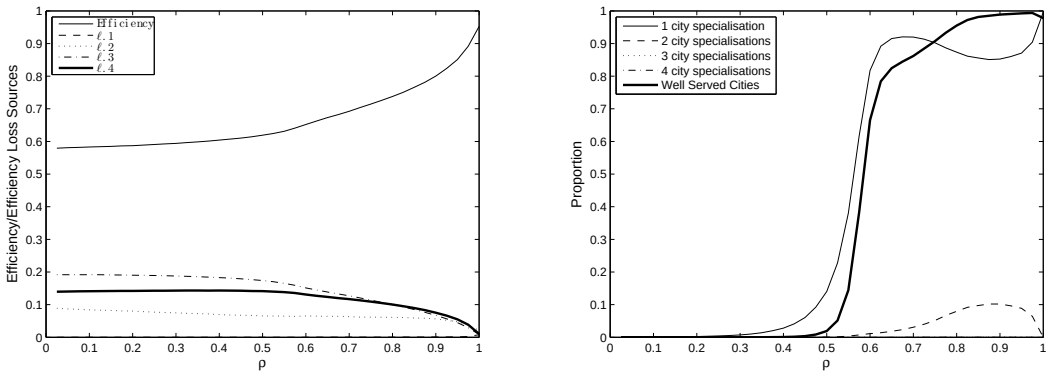
(a) (b)

Figure 4.1: Efficiency and loss sources (ta), and specialisation behaviour (b) for $R_{a/m} = 1$ and $\rho = 0.95$ over a single run of 500 iterations.

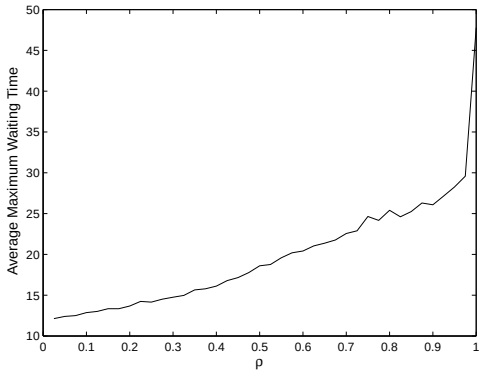
ratio at which all agents could take mail every iteration (no wasted resources). We also set $N_a = 5 \times 10^4$ and have shown in [26] that this is sufficient to neglect finite size effects. For the threshold function, we take $\lambda = 2$ and set $\theta_{min} = 0$ in order to minimise $\boldsymbol{\ell}.2$, and $\theta_{max} = 50$ sufficiently high to avoid most repeated changeovers. Finally, for the memory parameters, we take $\mu_a = 10$, $\mathcal{L}_a = 10$ and $\rho = 0.95$.

Figure 4.1 shows the performance of the algorithm over the course of a single run. We see that within the first 50 iterations the efficiency quickly tends to a high value while $\boldsymbol{\ell}.1$-$\boldsymbol{\ell}.4$ all take small values. This is followed by a slow increase in the efficiency to its asymptotic value which is mainly due to a corresponding decrease in $\boldsymbol{\ell}.2$. We see the reason behind this in the specialisation behaviour, with almost all cities being well-served by either singly or doubly specialised agents, fulfilling $\boldsymbol{c}.3$. Subsequently, some of the remaining unspecialised agents gain specialisation in a city being served by a doubly specialised agent, which then loses its second specialisation, This fulfils $\boldsymbol{c}.1$ and $\boldsymbol{c}.2$ and allows further increases in efficiency.

Figure 4.2 shows the influence of ρ on the efficiency. The efficiency is a monotonically increasing function of ρ, with the most marked increase taking place for $\rho > 0.5$. This increase can be explained by the specialisation behaviour, with city-specialisation starting to become prevalent at this point, as agents return often enough to cities for their average weights to increase. Although specialisation in up to 10 cities is in principle possible, we observe that most agents specialise in a single city. Double city-specialisations also occur for intermediately high values of ρ, but for $\rho \approx 1$ the chances of a specialised agent to visit another city become so small that

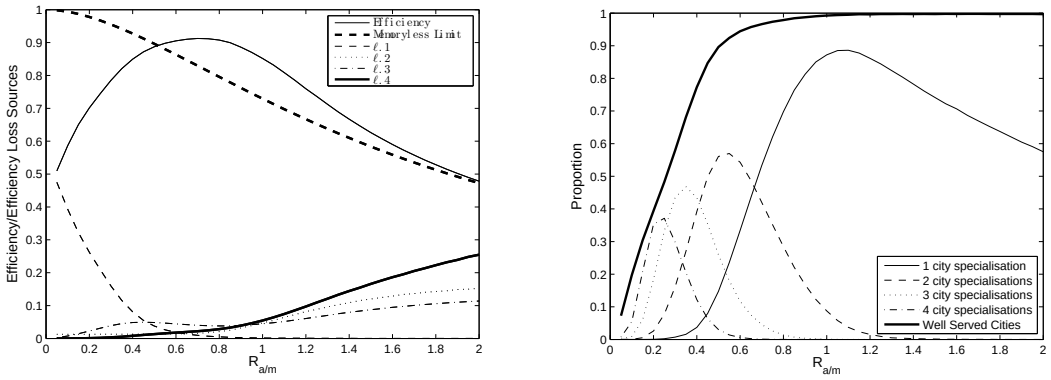


(a) (b)



(c)

Figure 4.2: Efficiency and loss sources (a), specialisation behaviour (b), and maximum waiting time (c) as a function of ρ for $R_{a/m} = 1$ averaged from iteration 401 to 500.



(a) (b)

Figure 4.3: Efficiency and loss sources (a), and specialisation behaviour (b) as a function of $R_{a/m}$ for $\rho = 0.95$ averaged from iteration 401 to 500.

very few doubly specialised agents emerge. We note, furthermore, that although the overall efficiency is maximised for $\rho = 1$, this is clearly not optimal for the fraction of well-served cities or the maximum waiting times. With increasing ρ a smaller and smaller fraction of agents visit cities at random, such that not well-served cities are less and less likely to be visited. The sharp increase after $\rho > 0.95$ is due to the fact that city-specialised agents not only never choose cities at random, but also stop specialising in multiple cities.

In figure 4.3 we compare efficiency with the upper bound on efficiency of any algorithm using random city choices, given by equation (3.23). At low values of $R_{a/m}$ high $\boldsymbol{\ell}.1$ leads to low efficiency compared to the limit as high average waiting times overwhelm the selectivity of the threshold function, leading to multiple changeovers. Note that this is a consequence of our choice of θ_{max}: higher values would lead to increased efficiency at low $R_{a/m}$ (due to lower $\boldsymbol{\ell}.1$) and decreased efficiency at high $R_{a/m}$ (due to higher $\boldsymbol{\ell}.2$, $\boldsymbol{\ell}.3$).

As $R_{a/m}$ increases, $\boldsymbol{\ell}.1$ decreases without much of an increase in $\boldsymbol{\ell}.2$-$\boldsymbol{\ell}.4$ as city specialisation becomes useful. Note that the average number of city specialisations becomes approximately $R_{a/m}^{-1}$ meaning that the agents are acting to serve approximately all the mail. This leads to most cities being well-served well before $R_{a/m} = 1$, fulfilling $\boldsymbol{c}.3$ and allowing efficiency to surpass the memoryless limit. The proportion of singly specialised agents continues to increase, fulfilling $\boldsymbol{c}.1$ and $\boldsymbol{c}.2$, but this does not lead to increased efficiency as it is inherently limited by the fact that there are less batches of mail than there are agents. Efficiency decreases as $\boldsymbol{\ell}.2$-$\boldsymbol{\ell}.4$ increase,

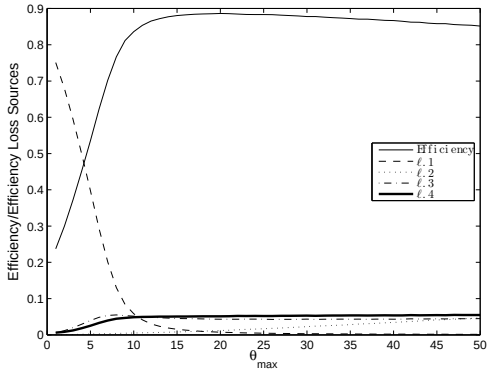


Figure 4.4: Efficiency and loss sources as a function of θ_{max} for $R_{a/m} = 1$ and $\rho = 0.95$ averaged averaged from iteration 401 to 500.

with these increases caused by both the lack of mail and the low average waiting times of the remaining batches. This means that less agents specialise in cities and efficiency approaches the memoryless limit, which also tends to the true upper limit $R_{a/m}^{-1}$ for high values.

Figure 4.4 shows the influence of θ_{max} on the efficiency. We observe that (too) low values for θ_{max} cause a high probability of changeovers (high $\boldsymbol{\ell}.1$). The efficiency increases sharply initially, peaking at approximately $\theta_{max} = 20$, after which it decreases due to increases in $\boldsymbol{\ell}.2$. These increases are due to higher average initial thresholds which lower the probability of initial mail uptake and hence specialisation. The behaviour is markedly different from that found in a memoryless system, for which much lower values of θ_{max} are optimal [26]. The increase in the optimal value of θ_{max} is due to the increased need to avoid changeovers. At a well-served city an agent that undergoes a changeover must not only cope with the penalty in processing time, but must now also compete for mail with another agent with whom it was previously cooperating.

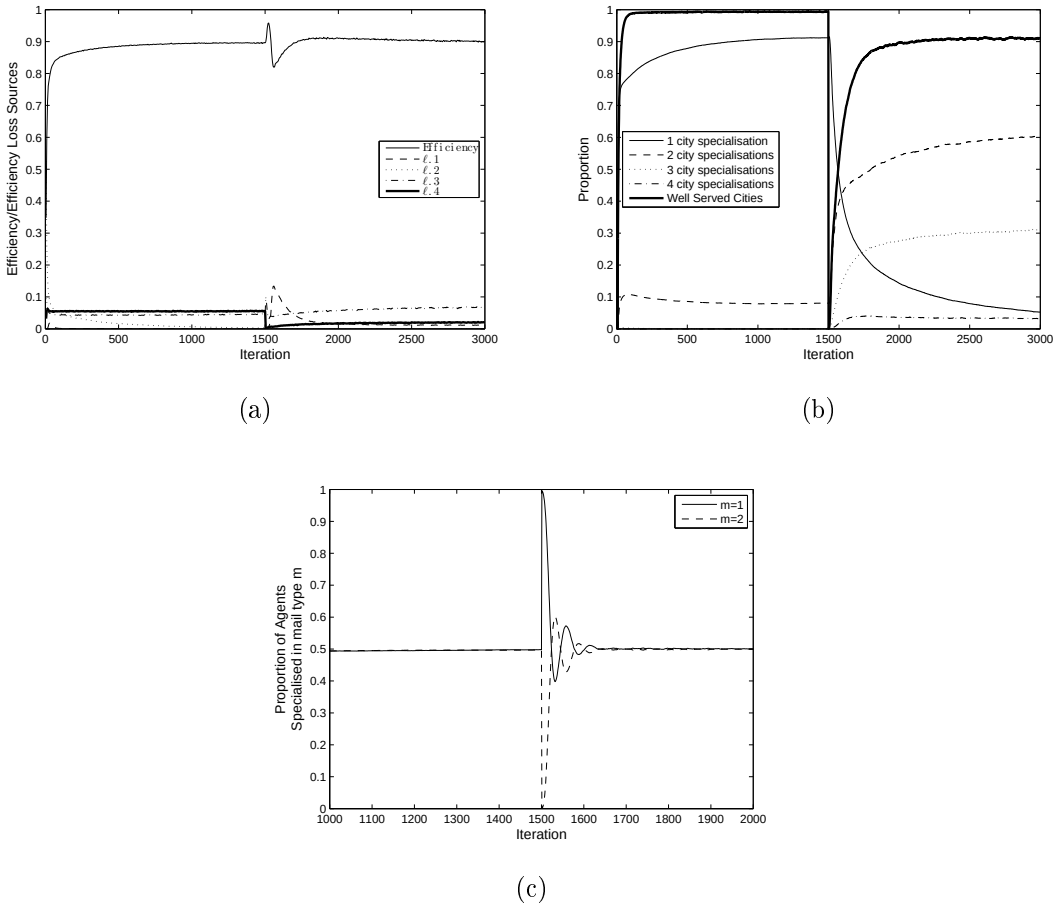


Figure 4.5: Efficiency and loss sources (a), city-specialisation behaviour (b) and mail-specialisation behaviour (c) for $\rho = 0.95$ a single run of 3000 iterations. After the first 1500 iterations, with $R_{a/m} = 1$, the agents specialised in mail type 2 were removed and the system was run for a further 1500 iterations with $R_{a/m} = 0.5$.

	Efficiency
Best Memoryless	0.645
Memoryless Limit	0.729
With Memory	0.953

Table 4.1: Comparison of the efficiency of the best threshold based memoryless algorithm with the theoretical limit for any memoryless algorithm and for the algorithm with memory ($\rho = 1$). Starting from uniform initial conditions and with no city-specialised agents, the efficiency is averaged over 500 iterations and the final efficiency is averaged over a subsequent 100 iterations.

In order to test the ability of the system to cope with abrupt changes, we allow it to converge over 1500 iterations. That half of the population with greatest specialisation (lowest thresholds) in mail type 2 is then removed and the remaining agents continue to process mail for 1500 more iterations. We see in figure 4.5 that after the removal of agents the efficiency initially increases, due to the increase in mail per agent, before changeovers cause an increase in $\boldsymbol{\ell}.1$ and corresponding sharp decrease in efficiency. Changeovers then decrease and efficiency steadily increases, reaching its long term value around 200 iterations after the removal of agents.

The (re-)specialisation behaviour highlights the causes behind the variations in efficiency. Note that since immediately after the removal no specialised agents in mail type 2 remain, the fraction of well-served cities decreases to 0. The increased waiting times in type 2 mail, combined with the fact that agents are generally specialised in a single city, causes over half of the population to switch mail-specialisation to type 2 within the first 50 iterations. As agents then gain additional city-specialisations, stable cooperation emerges and the proportion of each mail-specialisation returns to 0.5. This allows the re-emergence of well-served cities, decreasing $\boldsymbol{\ell}.1$ and allowing efficiency to return to a high level. Although the details differ, similar results are obtained if agents are randomly removed independently of their specialisations.

Table 4.1 compares the best actual efficiency for a fully optimised, hybrid threshold and market, memoryless algorithm [26], the theoretical upper limit of memoryless efficiency, and the best efficiency obtained in this chapter using SB memory with partially optimised parameters. Note that we use efficiency as the measure of performance and so take $\rho = 1$ even though this dramatically increases maximum

waiting times and is not optimal for other measures such as the fraction of well-served cities. We see that memory provides a large boost in efficiency. Note that the final efficiency of the only partially optimised SB memory system is at 98.6%, which is indeed very close to perfect efficiency.

4.3.1 Conclusions

In this chapter, we have introduced the SB model of agent memory as a solution to a problem of distributed task selection. The performance of this model has been investigated under the variation of key parameters and is close to that of the best centralised solution, while retaining the necessary conditions for good scalability such as a (very) limited information flow, localised decision making and relatively simple agents. In particular, the elimination of random city choices allows the system to exceed the theoretical upper limit on memoryless efficiency, an upper limit on the performance of the standard methods, by 35.3% after convergence and to obtain near perfect efficiency. This is partially due to emergent cooperation between the agents, which resolves the conflict between the need for agent flexibility, and the constraints of the model.

5 Premature Convergence in PSO

CONTENTS

In a static minimisation problem, convergence (unless guaranteed to the global minimum) serves no particular purpose. As long as the best point discovered so far is remembered, the actual distance to the current best is irrelevant. The main advantage of a converging PSO algorithm is the local search behaviour it generates, with effort switching from exploration of the search space to full exploitation of the region of the best current local minima. Once this region has been identified, however, the semi-random moves of PSO are far less efficient than gradient based methods [6]. As such, it seems wrong to sacrifice PSO's most effective feature: the efficient search of the global landscape, for inefficient fine-tuning of a solution which is probably worse than the global optimum. However, to discard local searching altogether is not desirable either, as it makes identification of the most promising solution very unreliable and requires a separate local search method, thus detracting from the simplicity of the algorithm.

In standard version of PSO, however, approaching convergence causes a breakdown in the global search capacity of the system. As mentioned previously, the search behaviour of PSO is driven by changes in the particles' local variables. Assuming that the parameters are set such that unbounded expansion is impossible (i.e. set according to the constriction coefficient method), a converging PSO will reach a point at which the set of possible future positions for all particles are either very close to their historic best position or are of greater cost. This means that little meaningful change to the local variables is possible and, therefore, that global search capacity has been lost. As discussed previously, if the combined benefits and chance of finding an improved minimum in the global space outweigh the benefits of optimising within the current best region we say that PSO has undergone **premature convergence**

Premature convergence can also occur in the PSO variants presented in the previous chapter, although the effect is mitigated by slower convergence speeds. However, convergence would be required to be infinitely slow in order to guarantee the avoidance of premature convergence. This is clearly impractical as we must work with finite resources and such a scheme effectively corresponds to no local search, which leads to the problems outlined above. As such it would be useful for PSO to be able to converge to a local minimum (gaining a good estimate of its true value) and then modify its behaviour, allowing it to regain global search capacity. Here we

describe a variant which allows this, the attractive and repulsive PSO, and introduce another, the dispersive PSO.

5.1 Attractive and Repulsive PSO

The **attractive and repulsive PSO** (ARPSO) algorithm (introduced by Riget and Vesterstrøm [60]) approaches this problem by allowing the algorithm to utilise two distinct search behaviours, or **phases**. While the algorithm is performing well, by either engaging in global search or attempting to exploit a local minimum, it is governed by the rules of some base PSO algorithm. If, however, the algorithm appears to have prematurely lost search capacity, and is attempting to fine-tune a well exploited local minima, it switches its behaviour in an attempt to regain it.

In general we cannot explicitly determine whether PSO has lost global search capacity until *all* search capacity is gone as doing so would require enough knowledge of our cost function to render local search pointless. However, a good indicator of search capacity is the swarm's **diversity** D, defined as

$$D = \frac{1}{n\ell} \sum_{i=1}^{n} |\vec{x}_i - \bar{x}| \tag{5.1}$$

where $\bar{x}$ is the centre of mass of the swarm and where $\ell = |\vec{b}^{max} - \vec{b}^{min}|$ is the length of the longest diagonal in the search space. We can see that for low diversity, particles occupy a relatively small region of the search space. While this can occur merely due to chance, it is far more likely that this is indicative of particles converging to a given point and so losing global search capacity. Conversely, high diversity shows that particles are spread widely in search space. This indicates that particles have a wide range of potential future positions and, therefore, a high chance that the system retains its global search capacity.

The ARPSO algorithm switches between its two search phases based on this diversity. Initially it proceeds according to its base PSO algorithm but if the diversity drops below a given threshold D_{low} it modifies its movement until sufficient diversity ($> D_{\text{high}}$) is regained. The search phases are:

- the **attractive** phase. This is the initial phase of the algorithm in which the swarm's movement is determined by some base PSO algorithm. While the

ARPSO algorithm was designed for use with the original PSO algorithm (i.e. movement according to equation (2.12)) there is no reason, in principle, why the attractive phase cannot use another PSO variant. If, during this phase, diversity decreases such that $D < D_{\text{low}}$, we switch to the repulsive phase of the algorithm.

- the **repulsive** phase. This phase attempts to regain search capacity by artificially increasing diversity. Equation (2.12) is modified by switching the sign of the social and cognitive terms, leading to *repulsion* from the attractor and neighbourhood historic best, and is given by

$$\vec{v}_i(t) = \eta\ \vec{v}_i(t-1) - c_1\ \vec{r}_1\ \otimes\ (\vec{a}_i(t) - \vec{x}_i(t)) - c_2\ \vec{r}_2\ \otimes\ (\hat{h}_i(t) - \vec{x}_i(t)) \quad (5.2)$$

  Repulsion continues until diversity has increased such that $D > D_{\text{high}}$, at which point the attractive phase is resumed.

This system does not allow the swarm to fully converge, sacrificing some local search capacity. However, the ability to continue searching even after a good candidate solution has been found means that premature convergence is avoided. Well chosen diversity thresholds should allow a reasonably close search of promising regions, while ensuring that global search capacity is not permanently lost.

5.2 Dispersive PSO

While the motivation behind ARPSO is sound, the algorithm itself has a number of issues which can improved upon

1. When introducing the ARPSO algorithm, we discussed the loss of search capacity that can result from clustering too closely around a historic best position. ARPSO takes low diversity to be the root cause of this loss and, therefore, directly increases diversity in order to regain search capacity. Recall, however, that a particle with fixed attractor and neighbourhood historic best oscillates about these points, eventually converging to a weighted combination [51; 70] and that this oscillation is effectively a PSO algorithm's local search procedure. This means that unless an ARPSO particle finds a better attractor in

another region through chance, adding diversity merely causes it to perform a local search on a larger area: it has no systematic way of finding other minima.

An ARPSO algorithm which has low enough diversity to trigger its repulsive phase is likely to have well optimised the minimum which its attractors are clustered round. If this minimum is of low cost then the regions containing even lower cost points (i.e. potential new attractors) may be very small. Thus the probability of a particle moving into one of these regions through chance is low, trapping the algorithm into a cycle of expansion and re-convergence to a single point.

2. The repulsive phase of the ARPSO algorithm introduces diversity but does not target good locations over poor ones. In fact, it is noted by Riget and Vesterstrøm [60] that so few improved solutions are found in this phase that it might be a more efficient use of resources not to calculate the fitness of particles until the attractive phase restarts. When combined with the previous issue, this makes finding an improved global optimum in high dimensional space becomes very difficult.

3. Low diversity causes the loss of global search capacity, but while it is a relatively good measure of proximity to a local optimum, it is not a direct one. If we have a region in which small increases in proximity to the minimum can lead to great improvements in fitness then we can only evaluate it by systematically *decreasing* diversity. As such, the algorithm should not stop its local search while it is still finding improved positions with reasonable frequency.

   Conversely, when a locally optimal solution is found, it may take some time for the diversity to decrease to below the minimum threshold. This means that the subsequent time steps not only waste function evaluations, but also unnecessarily decrease diversity. This effect is exacerbated in the case of PSO variants which slow diversity loss in order to better optimise multi-modal cost functions. This is problematic as these are exactly the type of functions that ARPSO is designed to solve, and it renders combinations of ARPSO with these variants sub-optimal.

In order to address these issues we propose the **dispersive particle swarm optimisation** (DiPSO) algorithm, which is based on principles not too dissimilar from the ARPSO algorithm. However, rather than relying on repulsion to enforce extra diversity, we allow attractors close to the current optimum to be forgotten once it has been sufficiently optimised. This allows swarm members to be attracted away (or disperse) from the current optimum towards other good positions, and prevents the cycle of re-convergence to a single good position.

Just as with the ARPSO algorithm, DiPSO acts on top of some base version of PSO. While the base algorithm is judged to be using its resources well DiPSO does not modify its behaviour in any way. However, once we are sufficiently confident that the base algorithm has lost search capacity, we create a **dispersal region** around the current optimum and define all attractors within this region to be *invalid*. Particles with *invalid* attractors use an alternative velocity update equation to the base PSO algorithm and the attractors themselves can be replaced by higher cost *valid* attractors, allowing the algorithm to regain some of its lost search capacity.

5.2.1 Details

An individual particle in a DiPSO population requires the full set of variables given in equation (2.5) and was our motivation for introducing the attractor variable. Note that when introducing the update of attractors, equation (2.4), we say that for standard PSO all positions are *valid* attractors and so a particle's attractor is identical to its historic best position. In the DiPSO algorithm, however, this is not the case.

When certain convergence conditions are met, a so-called **dispersal region** $\mathcal{R}_d(t)$ (analogous to a single entry tabu list in the tabu search algorithm [24; 25]) is created around a **dispersal centre** $\vec{c}_d(t)$. Local attractors are forbidden to lie within this region and any local attractors $\vec{a}_i(t) \in \mathcal{R}_d(t)$ are forgotten, by setting them to the null-vector $\vec{\emptyset}$. This vector is defined to have the property that $f(\vec{\emptyset}) > f(\vec{x}) \quad \forall \vec{x}$, so that it can be replaced by any valid position, and particles for which $\vec{a}_i (= \vec{\emptyset})$ are described as **dispersing**. Formally, when a dispersal region is created we update

particle i's attractor according to the following equation:

$$\vec{a}_i(t) = \begin{cases} \vec{a}_i(t) & \textbf{if } \vec{a}_i(t) \notin \mathcal{R}_d(t), \\ \vec{x}_i(t) & \textbf{else if } \vec{x}_i(t) \notin \mathcal{R}_d(t), \\ \vec{\emptyset} & \textbf{otherwise}. \end{cases} \tag{5.3}$$

Apart from when the dispersal region is initialised, the local attractors are updated identically to equation (2.4), although we explicitly define *valid* attractors as those falling outside the dispersal region.

$$\vec{a}_i(t+1) = \begin{cases} \vec{x}_i(t) & \textbf{if } \vec{x}_i(t) \notin \mathcal{R}_d(t), \textbf{ and } f(\vec{x}_i(t)) < f(\vec{a}_i(t)), \\ \vec{a}_i(t) & \textbf{otherwise}. \end{cases} \tag{5.4}$$

If the base PSO algorithm used is CLPSO, we also remove any exemplars inside the dispersal region, and reinitialise them in the next iteration that the particle has a valid local attractor. As the exemplar is the only point a particle is attracted to in the CLPSO algorithm, we wish to guarantee that it lies outside the dispersal region. As such, if a newly generated exemplar lies within $\mathcal{R}_d(t)$, we move it to the nearest boundary of $\mathcal{R}_d(t)$ (the details of this depend on the form of the dispersal region).

The forgetting mechanism combined with making certain positions invalid should allow DiPSO to find attractors in new regions of the fitness space. As these new attractors are unlikely to be locally optimal, improved attractors within these regions are likely to be found. As the search behaviour of PSO is driven by exactly these types of changes in remembered positions, DiPSO should be able to recover search capacity damaged by finding low cost positions.

The underlying assumption behind the search method of standard PSO is that good positions have characteristics in common. In particular it is assumed that low cost positions are proximate in the search space and, therefore, that the best place to search for the global minimum of a function is in the regions surrounding previously discovered positions which have the lowest cost, the historic bests. While this is not true for some complex problems, we are taking PSO as our base search method and, therefore, must ensure our algorithm is compatible with the basic functionality of PSO. Because of this we do not wish to completely disregard historic best positions, only to remove the absolute dependence on them. As such, we allow the particles to "socially" use the historic best positions of their neighbours[1], but only to "cognitively"

[1]Note that a particle can be a neighbour to itself.

use their own attractor.

The justification behind this choice is that we cause particles to forget their attractors when we believe the system has converged. At such a time we expect the set of historic best positions to be low diversity (as the converging PSO will tend to cluster around the global historic best) and more difficult to improve upon than the newly created attractors. Conversely we expect the set of attractors found after convergence to be of higher diversity and more liable to change as new positions are explored. As we want the remembered positions driving search behaviour to be both diverse and changing, and as socially sharing a variable cuts down the diversity of information used (due to selectivity), we choose to share the low-diversity, low-change historic bests and allow the attractors to be used individually.

We have defined two types of PSO such that they are compatible with DIPSO: standard PSO and CLPSO. Note that in the parts of the algorithms in which the usage of the particles' local variables is defined (the movement equation (2.12) for PSO and the exemplar update, algorithm 1, for CLPSO) there is a well defined balance between using local and neighbourhood information. This balance is independent of the number of neighbours, and can be simply weighted by adjusting either c_1 & c_2 in standard PSO, or by changing the learning probability, Pc_i, in CLPSO.

5.2.2 Movement

As dispersing particles lack the local information required by base PSO algorithms to define movement updates, we must introduce new rules governing their movement. These rules can be split into three cases.

- For *non-dispersing particles*, movement in the DiPSO algorithm is identical to that of its corresponding base algorithm.

- For *dispersing particles*, the local attractors $\vec{a}_i(=\vec{\emptyset})$ cannot be used to update the velocity. Neither should dispersing particles be attracted to $\hat{h}(t)$ which with high probability lies within the dispersal region. How dispersing particles update their velocity depends on whether their set of *valid* (i.e. non-dispersing neighbours) $\mathcal{V}_i = \{j \in \mathcal{N}_i \ : \ \vec{a}_j(t) \neq \vec{\emptyset}\}$ is empty or not.

- When all particles are dispersing, they are all repulsed by the dispersal centre

$\vec{c}_d(t)$. This is in order to prevent a deadlock situation in which all particles are stuck within the dispersal region.

Formally we can write this as

$$\vec{v}_i(t) = \begin{cases} \text{set according to base algorithm} & \textbf{if } \vec{a}_i(t) \neq \vec{\emptyset}, \\ \eta\vec{v}_i(t-1) + \phi\,\vec{r}_2 \otimes (\vec{\alpha}_i(t) - \vec{x}_i(t)) & \textbf{else if } \mathcal{V}_i \neq \emptyset, \\ \eta\vec{v}_i(t-1) - c_2\vec{r}_2 \otimes (\vec{c}_d(t) - \vec{x}_i(t)) & \textbf{otherwise}. \end{cases} \tag{5.5}$$

where η, c_1, $c_2 > 0$ and $\vec{r}_1$, $\vec{r}_2$ are chosen similarly to their counterparts in eq. (2.8). The **dispersal attractor** $\vec{\alpha}_i(t)$ is a stochastic combination of the above average normal local attractors, weighted by fitness:

$$\vec{\alpha}_i(t) = \frac{\sum_{j\in\mathcal{V}_i^+} r_j(f(\vec{a}_j(t)) - \overline{f_{\mathcal{V}_i}})\ \vec{a}_j(t)}{\sum_{j\in\mathcal{V}_i^+} r_j(f(\vec{a}_j(t)) - \overline{f_{\mathcal{V}_i}})} \tag{5.6}$$

where $\overline{f_{\mathcal{V}_i}} = \frac{1}{|\mathcal{V}_i|}\sum_{j\in\mathcal{V}_i} f(\vec{a}_j(t))$ is the average fitness of particle i's valid neighbours, and $\mathcal{V}_i^+ = \{j \in \mathcal{V}_i \ : \ f(\vec{a}_j(t)) \geq \overline{f_{\mathcal{V}_i}}\}$ is the set of valid neighbours of greater than average fitness. The form of this attractor can be thought of as a special case of the informed attractor described in the **fully informed PSO (FIPSO)** algorithm algorithm introduced by Mendes et al. [48]. Note that particles can move within $\mathcal{R}_d(t)$, but cannot set their local attractor in it.

This stochastic combination of information from good, valid, neighbours should allow dispersing particles to engage in some sort of principled search behaviour, even before it regains access to its own cognitive function.

5.2.3 Convergence Conditions

As discussed previously, basing detection of convergence entirely on swarm diversity can result in under-exploitation of a local minimum (if low-diversity improvements are relatively) or wasted function evaluations (if we are using a PSO variant which slows diversity loss). As, to preserve the generality of our algorithm, we do not want to assume knowledge of the gradient of our fitness space we cannot directly measure our solution quality. Therefore, we have implemented a compromise designed to avoid the extreme consequences of a convergence measure which is entirely diversity dependent. Our measure is diversity based, but is buffered by a time dependent element.

In order to define our convergence conditions we also need to define the concept of algorithmic stationarity. This concept is driven by the algorithm's current best attractor. Note that as $\vec{a}_i(t)$ only contains the personal best position of particle *since* its last dispersal, we define the **current best attractor** as:

$$\hat{a}(t) = \underset{\vec{a}_i(t)}{\operatorname{argmin}} f(\vec{a}_i(t)), \quad i = 1, ..., n \tag{5.7}$$

We use this to introduce the concept of *algorithm* stationarity, similar to *particle* stationarity as defined for CLPSO. The system is said to have been **stationary** for τ time steps if $\hat{a}(t) \neq \vec{\emptyset}$ and τ is the largest number for which:

$$f(\hat{a}(t-s)) = f(\hat{a}(t)), \forall s < \tau. \tag{5.8}$$

We now define two critical times for convergence: t_D and t_{abs}. Given the system has been stationary for τ time-steps, we say that it has **converged** if one the following conditions is met:

1. $\tau > t_D$ and the diversity as defined in equation 5.1 is is such that $D < D_{\text{low}}$.
2. $\tau = t_{\text{abs}}$

When either of these conditions is satisfied, we initialise the dispersal region $\mathcal{R}_d(t)$ and remove local attractors which lie inside it as discussed in section 5.2.1. A newly created dispersal region supplants existing regions.

Note that, for appropriate parameter values, t_D should provide a buffer against the early creation of a dispersal region in situations when local improvement is still viable. Likewise, t_{abs} should protect against wasted function evaluations in slowly converging variants as well as providing a fail-safe against the algorithm becoming stuck without converging. Clearly, in order for our diversity condition to be meaningful, we require $t_D < t_{\text{abs}}$.

5.2.4 Dispersal Region

Now we discuss the choice and behaviour of the dispersal region $\mathcal{R}_d(t)$. As the dispersal region has the ability to stop some areas of the search space being exploited, its size, shape and permanence are critical to the behaviour of the DiPSO algorithm. If the region is too small, local attractors can still cluster around the dispersal point.

If it is too large, we risk excluding good minima within the dispersal region. As the properties of an optimal dispersal region are entirely problem dependent, we propose a simple heuristic for setting an appropriately large dispersal region while ensuring that good minima are not permanently excluded from the algorithms search region.

The dispersal region is characterised by two variables:

- The **dispersal centre** $\vec{c_d}(t)$, a point at which we think an exploited local minimum lies.
- The **dispersal radius** $r_d(t)$ which along with ℓ, the length of longest diagonal in the search space, defines the size of the dispersal region.

Given these variables, we say that a point $\vec{x}$ lies inside the dispersal region if

$$|\vec{c_d}(t) - \vec{x}| < \ell \cdot r_d(t) \tag{5.9}$$

where ℓ is used to scale dispersal radius size with the size and dimension of the search space.

When DiPSO's convergence conditions are met, $\vec{c_d}(t)$ is set to $\hat{a}(t)$ which, as the point our algorithm has converged to, should be a good estimate for the location of a local minima. Note that using a static dispersal radius r_d could critically affect the performance of the algorithm. Therefore, we initially set the dispersal radius to some relatively high value r_d^{max} and linearly reduce it to 0 over the course of t_d time steps. This should stop any initial clustering around $\vec{c_d}(t)$, while allowing minima inside the initial dispersal region to be found during later iterations.

While our algorithm now has many additional parameters, we do not consider this to be a great disadvantage as we are seeking a robust algorithm, as opposed to an optimal one. While it would be possible to set the parameters such that the algorithm performs very badly, we give the following heuristic guidelines which we believe will lead to good robustness.

- The dispersal conditions should be difficult to meet. While spending additional time in a converged state is costly in terms of function evaluations, an algorithm which is frequently interrupted during promising searches will not tend to find good solutions.

- The dispersal region should initially be large, and should reduce its size slowly. A large initial dispersal region should allow particles to find attractors outside the basin of attraction of the current historic best. A slow reduction in dispersal radius gives particles a chance to find solutions which are locally optimised to some degree before high quality solutions close to the current historic best become valid, encouraging good swarm diversity after convergence.

While these guidelines will not give us the fastest possible version of DiPSO, the purpose of DiPSO is to be robust when other PSO methods are not. Therefore we allow some loss of speed in order to ensure good robustness.

Note that we make no claim as to the optimality of this scheme, but finding an optimal, general, scheme is beyond the scope of this thesis. Also, it is worth noting that with some knowledge of the fitness landscape DiPSO is working in, it should be possible to devise a dispersal region, specific to the problem, in order to improve performance.

5.3 Results

As we cannot test our algorithm on all problems it might reasonably be used on, we require a set of problems which can determine its behaviour in a range of circumstances. Because the results of a PSO algorithm over a range of runs on a difficult problem are highly chaotic, we will conduct a large number (10^4) of repeat trials per algorithm, per function in order to gain a reasonable estimate on their performances. This, combined with practical limits on computing time mean that we carefully chose a small range of functions on which to conduct our tests.

The functions used by Potter and Jong [54] contain a uni-modal function, the Rosenbrock function, three multi-modal functions with different properties, the Ackley, Griewank and Rastrigin functions, and one multi-funnelled function[2], the Schwefel function. To this we shall add a simple uni-modal problem, the well known sphere function, along with a multi-funnelled function which evolutionary algorithms tend to fail to fully optimise, the Rana function [72]. We feel this problem set is suitable for a preliminary test of the performance of DiPSO.

[2]Multi-funnelled functions are those multi-modal functions in which jumping to successively better local minima can lead an algorithm *away* from the global minimum.

The function definitions, along with their bounds, are given below. Where necessary, they have been adjusted to give a minimum value of 0.

The **Rosenbrock function**, defined as

$$\sum_{i=1}^{d-1}(1-x_i)^2 + 100(x_{i+1}-x_i^2)^2, \qquad \vec{x} \in [-5.12, 5.12]^d$$

is a uni-modal function characterised by a shallow, curved, valley leading to the global minimum. This valley is easy to find, but due to its low gradient may cause algorithms to slow down and become stuck.

The **sphere function**, defined as

$$\sum_{i=1}^{d} x_i^2, \qquad \vec{x} \in [-5.12, 5.12]^d$$

is a very simple uni-modal function in which cost decreases directly with distance to the global minimum. It is useful for determining algorithms' speed when exploiting a minimum.

The **Ackley function**, defined as

$$20 + e - 20\exp\left(-0.2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{d}\cos(2\pi x_i)\right),$$
$$\vec{x} \in [-32.768, 32.768]^d$$

is a function with a uni-modal trend, given by the first exponential term. The second exponential term introduces local fluctuations, making the function multimodal.

The **Griewank function**, defined as

$$1 + \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right), \qquad \vec{x} \in [-600, 600]^d$$

is a function similar to the sphere function but with multi-modality introduced due to sinusoidal variations. Note that this function is non-separable, meaning that algorithms which work on a variable-by-variable basis will break down.

The **Rastrigin function**, defined as

$$10 \cdot d + \sum_{i=1}^{d-1} x_i^2 - 10\cos(2\pi x_i), \qquad \vec{x} \in [-5.12, 5.12]^d$$

is a multimodal function characterised by regularly spaced minima, decreasing in cost towards global minimum. If an algorithm can efficiently move between neighbouring global minimum, it can optimise this function by repeatedly moving to a lower cost neighbouring minimum.

The **Rana function**, defined as

$$512.7531 + \sum_{i=1}^{d} \left(\frac{x_i \sin(\alpha_i)\cos(\beta_i) + (x_j + 1)\sin(\beta_i)\cos(\alpha_i)}{d} \right), \qquad \vec{x} \in [-520, 520]^d$$

where $j = i + 1 \mod d$, $\alpha_i = \sqrt{|1 - x_i + x_j|}$ and $\beta_i = \sqrt{|1 + x_i + x_j|}$, is a multi-funnelled function which is non-separable and highly multimodal. It is a difficult problem for many evolutionary algorithms, and we use it as a test case for problems our algorithms struggle with.

The **Schwefel function**, defined as

$$418.9829 \cdot d - \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|}), \qquad \vec{x} \in [-500, 500]^d$$

is a multi-funnelled function with good local minima far from the global minimum. It is a useful test of an algorithm's ability to transition between good minima without the need for intermediate steps.

The main aim of this section is to investigate whether the DiPSO algorithm can systematically improve the robustness of its base PSO algorithms, and to compare this improvement with its nearest competitor, the ARPSO algorithm. In order to do this, we test both the standard PSO and CLPSO with no modification, with ARPSO and with DiPSO on 10, 30 and 100 dimensional versions of the above problem set. In order to better understand how these improvements come about we also investigate the performance of DiPSO as a function of time, both on average, and over the course of a single run. Note that, for notational convenience, we will use **C-ARPSO** to denote an ARPSO algorithm acting on top of the CLPSO algorithm with S-ARPSO meaning that standard PSO was used as a base algorithm. We also use a similar convention for DiPSO.

Results are given both in terms of performance, the average cost of the best solution , and robustness, the fraction of trials in which the algorithm has found a solution of acceptable quality. These results are taken after a set number of function evaluations, $1 \cdot 10^6$ unless otherwise specified. An **acceptable solution** is defined to be one with cost less than 0.1, except in the case of the Rana function, in which we take the threshold to be 75 due to its high degree of difficulty. We also measure how quickly, on an average run in which an acceptable solution was found, the acceptability threshold was passed.

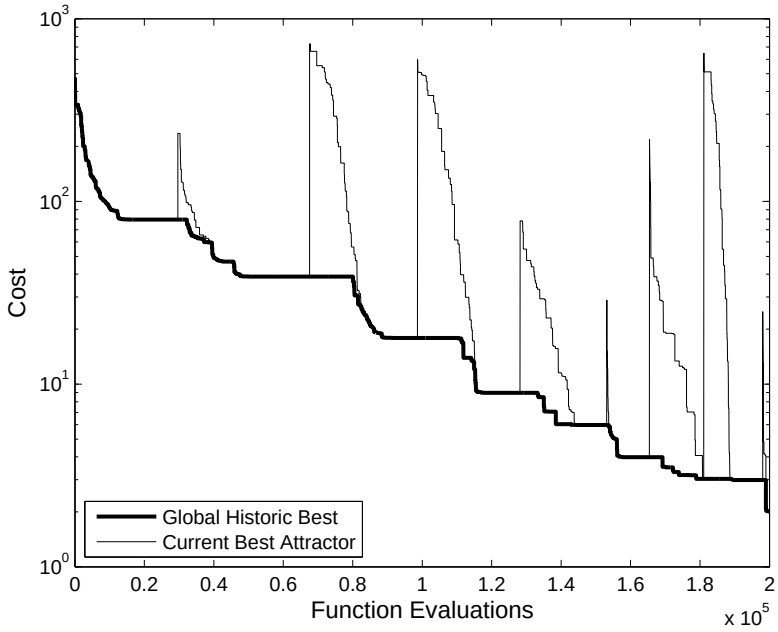


Figure 5.1: A single run of the DiPSO algorithm on the 30-dimensional Schwefel function.

5.3.1 Behaviour

Figure 5.1 is illustrative of the behaviour of the DiPSO algorithm over a single run. We see that following a period of stationarity in the global historic best cost, the *current* best cost increases as the algorithm disperses. This increase in cost can be dramatic, but allows the swarm to systematically improve and find a new lower cost position.

We can also see that, sometimes (e.g the first dispersal period shown), the global historic best cost improves during a dispersal period *ahead* of the increase in current best cost. This is possible because the swarm can move within the dispersal region and freely update their historic best positions. Often these new positions are not fully optimised until the dispersal radius decreases enough to allow attractors to be set in this region. Once this occurs a full local search of the area can take place, allowing efficient exploitation of the new minimum.

Figure 5.2 illustrates the average behaviour of DiPSO, compared to the other PSO variants. We can see that when the other algorithms get stuck at a reasonable level of average cost, DiPSO can still be capable of making improvements. The mechanism behind this can be seen (from the figure 5.2 (b and d)) to be its robustness. In other variants, those swarms finding acceptable quality solutions may do so

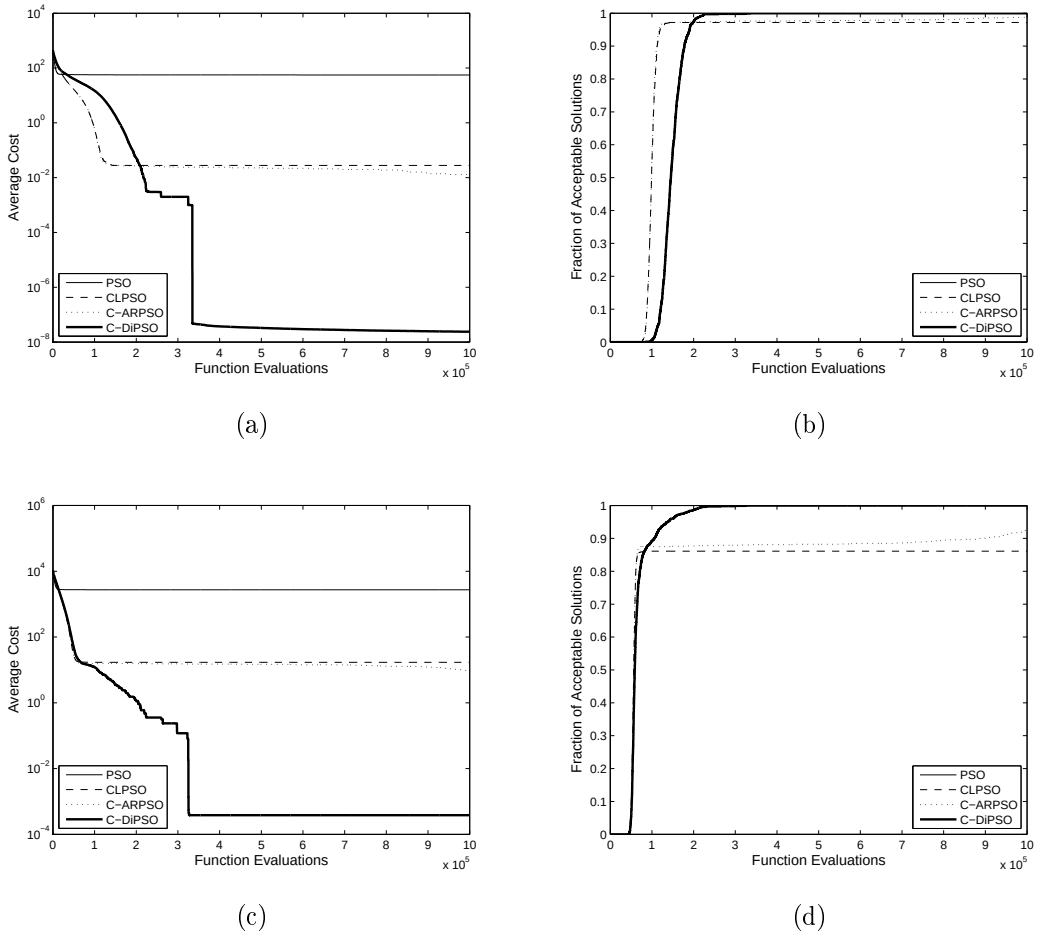


Figure 5.2: Average performance (a and c) and robustness (b and d) of our PSO algorithms when applied to the Rastrigin function (a and b) and to the Schwefel function (c and d), both 30-dimensional.

more quickly than DiPSO. However, those swarms which do not find such a solution quickly may never do so (or in the case of the C-ARPSO algorithm applied to the schwefel function, may do so very slowly). The DiPSO algorithm, conversely, tends to find acceptable solutions at a steady, if slightly slower rate.

Note that if another PSO variant had been able to find the global minimum of the Rastrigin function with certainty, then the slowing effect of the dispersal regions in DiPSO would render its performance inferior. This does happen in some tests, and we will highlight these cases in the following section.

5.3.2 Robustness

In order to systematically compare DiPSO to the other algorithms, we compare their performance and robustness across all test functions in 10, 30 and 100 dimensions.

Similarly to in figure 5.2, the average cost is largely driven by robustness. This is because those test runs of an algorithm which do not converge to the global minimum account for a large proportion of the average cost and so a slightly lower robustness may lead to a greatly increased cost. As such, we only show details of robustness in the main body of the text. For details of the average cost, see appendix C.

when giving robustness results for an algorithm, we give both the fraction of runs in which it found an acceptable quality solutions and (in brackets) **solution time**: the average time at which it found such solutions. Runs in which no acceptable solutions were found are discounted for the purpose of this average. When deciding which algorithm performed best on a particular problem (those solutions highlighted in bold), we first look at which is the most robust. If there are number of equally good algorithms in terms of robustness, then the one with the lowest solution time is deemed to be best. Note that as we are looking at *improvements* in robustness, we will tend to compare algorithms with the same base method for a given trial.

Table 5.1 shows the robustness of the PSO variants on a 10-dimensional problem set. We can see that, on a given problem and for a fixed base, DiPSO is always at least as robust as the best alternative. However because the other PSO variants are also very robust on this problem set there is usually a variant which beats it in terms of solution time. ARPSO also tends to improve the robustness of its base algorithms, but not to such a large degree. In general, however, it shows improved speed when compare to DiPSO variants of equal robustness.

Table 5.2 shows the robustness of the PSO variants on a 30-dimensional problem set. A similar general pattern is seen to that on the lower dimensional problem set, with DiPSO at least as robust as other methods. The one exception is the case of the Griewank function when using standard PSO as a base, in which S-ARPSO is more robust than S-DiPSO. As this difference is as a result of one trial out of 10^4, however, it is quite possible that this is because of a statistical fluctuation. When other variants reach the robustness of DiPSO, they also tend to have a lower solution time. However, the robustness advantage DiPSO is highlighted by this more difficult problem set, particularly when standard PSO is used as a base.

We can see from table 5.3 that when the base method is capable of solving the problem with reasonable robustness (sphere function, Ackley and Griewank functions with CLPSO as base algorithm) DiPSO retains its high robustness, and its

Table 5.1: Robustness of the PSO algorithms in 10 dimensions.

Function	Base	No Convergence	ARPSO	DiPSO
Rosenbrock	Standard	0.855 (54869.8)	0.913 (80055.8)	**1** (82197.4)
	CLPSO	0.942 (58430.8)	0.942 (58430.8)	0.997 (196593)
Sphere	Standard	**1** (1681.52)	**1** (**1671.94**)	**1** (1681.42)
	CLPSO	**1** (4895.46)	**1** (4925.8)	**1** (4913.32)
Ackley	Standard	0.995 (3753.93)	**1** (**3785.36**)	**1** (3843.1)
	CLPSO	**1** (13494.7)	**1** (13596.8)	**1** (13905.6)
Griewank	Standard	0.936 (12769.9)	0.981 (20110.4)	0.991 (12677.6)
	CLPSO	**1** (**14008.4**)	**1** (14032.4)	**1** (31801.6)
Rastrigin	Standard	0.016 (163862)	0.905 (195868)	**1** (54061.8)
	CLPSO	0.999 (20821.5)	**1** (**20944.3**)	**1** (27451.4)
Rana	Standard	0.426 (57535.5)	0.532 (114672)	0.997 (68859.3)
	CLPSO	**1** (**41590.6**)	**1** (**41590.6**)	**1** (65526.3)
Schwefel	Standard	0.098 (28980.4)	0.966 (97375.4)	0.998 (50658.5)
	CLPSO	0.982 (15100.9)	**1** (19457.2)	**1** (**15719**)

Table 5.2: Robustness of the PSO algorithms in 30 dimensions

Function	Base	No Convergence	ARPSO	DiPSO
Rosenbrock	Standard	0.748 (264167)	0.82 (318706)	**0.97** (348889)
	CLPSO	0.82 (246464)	0.82 (246464)	0.954 (443845)
Sphere	Standard	**1** (6474.66)	**1** (6449.82)	**1** (**6443.16**)
	CLPSO	**1** (17958)	**1** (17947)	**1** (18098.8)
Ackley	Standard	0.758 (12070.6)	**1** (**16892.6**)	**1** (25539.6)
	CLPSO	**1** (39571)	**1** (39695.3)	**1** (44788.1)
Griewank	Standard	0.999 (11867.1)	**1** (**11903.5**)	0.999 (12482.4)
	CLPSO	**1** (35717.6)	**1** (35564)	**1** (61106)
Rastrigin	Standard	0	0.546 (658922)	**1** (252237)
	CLPSO	0.972 (99323.5)	0.987 (106809)	**1**(**148216**)
Rana	Standard	0	0.024 (675895)	0.46 (662041)
	CLPSO	**1** (**335087**)	**1** (**335087**)	**1** (516287)
Schwefel	Standard	0	0.146 (860858)	0.666 (463305)
	CLPSO	0.861 (55483.6)	0.924 (94055.5)	**1** (69263.3)

Table 5.3: Robustness of the PSO algorithms in 100 dimensions

Function	Base	Standard	ARPSO	DIPSO
Rosenbrock	Standard	0.001 (382500)	**0.014** (560456)	0.003 (624440)
	CLPSO	0	0	0
Sphere	Standard	**1** (35347.3)	**1** (35388.5)	**1** (**35264.2**)
	CLPSO	**1** (60452.1)	**1** (60520.4)	**1** (63959.8)
Ackley	Standard	0	**1** (391179)	0.025 (763777)
	CLPSO	0.825 (114199)	**1** (**135811**)	**1** (156790)
Griewank	Standard	0.708 (54854.5)	0.79 (76797.4)	0.806 (86093.5)
	CLPSO	**1** (**99409.4**)	**1** (99592.5)	**1** (147326)
Rastrigin	Standard	0	0	0
	CLPSO	**0.029** (369641)	0.026 (431492)	0.021 (854183)
Rana	Standard	0	0	0
	CLPSO	0	0	**0.44** (960146)
Schwefel	Standard	0	0	0
	CLPSO	0.082 (214824)	0.106 (232762)	**0.823** (513781)

corresponding cost in speed. On some of the other problems, however, it doesn't systematically lead to increased robustness and is even comprehensively outperformed by S-ARPSO on the Ackley function. Also, while C-DiPSO is significantly more robust than other methods on the multi-funnelled problems, its robustness is less than it achieved in lower dimensions.

An indication of the reasons behind this lie in the solution times. Many of them are close to the limit on function evaluations ($1 \cdot 10^6$), especially when it is considered that they are an *average* solution time. This leads us to believe that the algorithms haven't finished improving, and require more time to solve the problem. This is not entirely surprising, due to the "curse of dimensionality" [35]. To counteract this effect, we will look at the results after a longer run of $(1 \cdot 10^7)$ function evaluations. For clarity, we will remove those problems which were already solved with high robustness.

We can see from table 5.4 that this extra time allows C-DiPSO to achieve close to perfect robustness on the Rastrigin, Rana and Schwefel functions while other methods struggle significantly. S-ARPSO also shows some improvement on these problems, but of a much smaller scale. Also its low solution times relative to the limit on function evaluations give us no indication that it would continue to improve, at a reasonable rate, if given more time. All algorithms show a small improvement on the Griewank function and S-DiPSO also becomes able to solve the Ackley problem

Table 5.4: Robustness of the PSO algorithms in 100 dimensions after 10^7 function evaluations

Function	Base	Standard	ARPSO	DIPSO
Rosenbrock	Standard	0.337 (990135)	**0.557** ($1.4739 \cdot 10^6$)	0.494 ($1.13677 \cdot 10^6$)
	CLPSO	0.02 ($1.58313 \cdot 10^6$)	0.014 ($1.75299 \cdot 10^6$)	0.022 ($1.63784 \cdot 10^6$)
Ackley	Standard	0	**1** (430056)	0.999 (935048)
Griewank	Standard	0.758 (59608.9)	**0.863**(**146884**)	0.86 (169278)
Rastrigin	Standard	0	0.001 ($2.94862 \cdot 10^6$)	0.672 ($1.63739 \cdot 10^6$)
	CLPSO	0.071 (715795)	0.137 ($1.5279 \cdot 10^6$)	**0.998** ($1.03502 \cdot 10^6$)
Rana	Standard	0	0	0
	CLPSO	0.127 ($2.83609 \cdot 10^6$)	0.127 ($2.83609 \cdot 10^6$)	**1** ($2.28999 \cdot 10^6$)
Schwefel	Standard	0	0	0
	CLPSO	0.127 (250641)	0.148 (724339)	**0.999** (669800)

with close to perfect robustness, a significant increase. Finally, on the Rosenbrock function, designed to cause problems with the speed of convergence, the additional time gives algorithms based on standard PSO the ability to solve the problem a reasonable fraction of the time, with S-ARPSO being the most robust.

5.3.3 Conclusions

In this chapter we have analysed the ARPSO algorithm with respect to its ability to recover search capacity following premature convergence in PSO. To address the issues identified we have introduced a new PSO variant, dispersive PSO, which can maintain good search performance under normal conditions before switching behaviour when the system has converged. The performance of DiPSO has been analysed in comparison to other PSO variants on a range of standard test functions with respect to robustness and average cost. We have seen that DiPSO is almost always more robust than other PSO variants and that this is particularly the case in difficult, high dimensional problems. We have also seen that DiPSO's ability to re-converge to the global optimum can lead to it having a low average cost more often than can be explained by robustness. These advantages, however, have been observed to come at a cost to speed in problems which standard PSO variants are capable of solving robustly.

An important question remaining is whether DiPSO is sensitive to parameter choice. Initial studies seem to back up our heuristic guidelines in section 5.2.4. The fact that the same parameters lead to good performance on a range of test functions also supports this. However, a more in depth investigation of the effect of parameters

on the DiPSO algorithm is necessary.

6 Summary & Future Work

CONTENTS

6.1 Summary

In this thesis, we have studied an agent based model for distributed mail retrieval. The efficiency and flexibility have been investigated both in static and dynamic environments. and with respect to catastrophic breakdowns of agents. We have introduced new rules for mail selection and specialisation and have used a evolutionary algorithm to optimise these further. We have shown that some of the new rules have improved performance compared to existing ones. The best ones give increased efficiency by 25.5% in a static, and 24.3% in a dynamic environment, compared to a method (VRT) which already outperformed a variety of other algorithms [56].

We have studied both nature inspired and market based algorithms for distributed task allocation applied to a problem of mail processing. We have investigated the algorithms' ability to cope with load and their adaptability. In particular, we found that nature inspired, threshold based algorithms have a far higher robustness to high load than the market based algorithms. Market based algorithms were found to be quicker to adapt to system changes than the threshold based algorithms, although this only translated into a small performance difference compared to the optimised SO algorithm.

We have identified the various loss sources, and have demonstrated that the random choice of cities to visit by the agents forms the main limitation on the maximal attainable efficiency. We have derived this limit theoretically. We also investigated the absolute performance of the algorithms in relation to this limit and, to that end, introduced a new hybrid approach and used a particle swarm optimisation algorithm to find good parameter sets for our algorithms. These algorithms, with optimised parameters, give us increased efficiency in a low load setting of 48.2% in a static, and 61.3% in a dynamic environment, compared to a method (VRT) which already outperformed a variety of other algorithms [56; 57]. In a high load setting these figures are 31.3% in a static, and 32.4% in a dynamic environment.

We have also introduced the SB model of agent memory as a solution to a problem of distributed task selection. The performance of this model has been investigated under the variation of key parameters and is close to that of the best centralised solution, while retaining the necessary conditions for good scalability such as a (very) limited information flow, localised decision making and relatively simple agents. In

particular, the elimination of random city choices allows the system to exceed the theoretical upper limit on memoryless efficiency, an upper limit on the performance of the standard methods, by 35.3% after convergence and to obtain near perfect efficiency. This is partially due to emergent cooperation between the agents, which resolves the conflict between the need for agent flexibility, and the constraints of the model.

Finally, we have analysed the ARPSO algorithm with respect to its ability to recover search capacity following premature convergence in PSO. To address the issues identified we have introduced a new PSO variant, dispersive PSO, which can maintain good search performance under normal conditions before switching behaviour when the system has converged. The performance of DiPSO has been analysed in comparison to other PSO variants on a range of standard test functions with respect to robustness and average cost. We have seen that DiPSO is almost always more robust than other PSO variants and that this is particularly the case in difficult, high dimensional problems. We have also seen that DiPSO's ability to re-converge to the global optimum can lead to it having a low average cost more often than can be explained by robustness. These advantages, however, have been observed to come at a cost to speed in problems which standard PSO variants are capable of solving robustly.

6.2 Future Work

While our task allocation algorithms are well studied and have been shown to be applicable to various problems, here we have made several modelling choices which may have affected the performance of our algorithms:

- *Local task sites* - In the current model tasks are clustered within distinct, identifiable locations. Information about the state of the tasks at a location can only be identified by visiting it. If tasks were instead distributed within an area of which an agent could gain some local overview then the agent could avoid visiting areas in which all mail had already been taken. Cases with a lack of memory would, however, still lead to unbalanced numbers of agents within different areas resulting in competition for mail.

- *Geometry* - We have assumed that each agent's processing centre is equidistant from each city and, therefore, that the time taken to visit a city and collect a batch of mail is independent of the city chosen. In a more realistic model, agents should take longer to travel to some cities than others. This should actually lead to an additional benefit of memory, as memories in distant cities should decay faster than an agent can build them up, driving agents to visit local cities and decreasing average travel time.

- *Static environment* - When testing the memory based algorithm, mail is produced constantly and uniformly at all cities neglecting the dynamic environment which we studied in other cases. The current system of memory does not differentiate between cities visited recently and those visited a longer time ago, which have a greater chance to have produced new mail batches. Building some repression of memory in recently visited cities into the model should allow agents to specialise in multiple cities in a more efficient way and make it more applicable to dynamic problems.

- *Breakdowns* - While we have seen that the system can adjust to compensate for the breakdown of agents, the memory based algorithm cannot quickly readjust after breakdowns in cities. The stability of established city-specialisations means that agents will return to cities long after they have ceased to produce mail. As for dynamic environments, a way to solve this problem would be to build some time dependence into the model, with memories decaying faster when not reinforced regularly.

Finally, the performance of our algorithms may still be limited by our choice of the functional forms of the algorithms and bidding functions, as such it would also be interesting to apply genetic programming in which agents are allowed to develop their strategies freely. Also, the design of a MB protocol which allows a greater degree of selectivity in accepting or rejecting tasks would be of great advantage in a high load setting. Our hybrid algorithm is a possible approach to this problem, however, it requires a separate adjustment of parameters between both high and low load settings and this needs to be addressed for it to be fully effective.

With regards to the DiPSO algorithm, we have also made a number of choices in our design process which may have affected our algorithms. Firstly, while we have

initial evidence to support our heuristic guidelines for setting DiPSO's parameters, it that we fully validate them. Secondly, our choices of the shape and persistence of the dispersal region, along with our convergence conditions, may be vital to performance. As we have chosen these features based on intuition rather than through thorough empirical or theoretical research there may be scope for large performance improvements, particularly if optimal behaviour can be designed for different problem classes. We also believe that their is scope for improvement on current base PSO methods when applied to non-separable and discrete functions, and an investigation is currently in progress.

Bibliography

[1] C. Anderson and F. Ratnieks. Task partitioning in insect societies. i. effect of colony size on queueing delay and colony ergonomic efficiency. *The American Naturalist*, 154(5), 1999.

[2] C. Anderson and F. Ratnieks. Task partitioning in insect societies. ii. use of queueing delay information in recruitment. *The American Naturalist*, 154(5), 1999.

[3] S. N. Beshers and J. H. Fewell. Models of division of labor in social insects. *Annual Review of Entomology*, 46:413–440, 2001.

[4] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing*, 1(1):3–52, March 2002. ISSN 1567-7818. doi: 10.1023/A:1015059928466. URL `http://dx.doi.org/10.1023/A:1015059928466`.

[5] E. L. Bienenestock, L. N. Cooper, and P. W. Munro. Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Nuroscience*, 2(1):32–48, 1982.

[6] H.K. Birru, K. Chellapilla, and S.S. Rao. Local search operators in fast evolutionary programming. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, pages –1513 Vol. 2, 1999. doi: 10.1109/CEC.1999.782662.

[7] E. Bonabeau, A. Sobkowski, G. Theraulaz, and J.-L. Deneubourg. Adaptive task allocation inspired by a model of division of labor in social insects. In *Biocomputing and Emergent Computation*, pages 36–45, 1997.

[8] E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg. Fixed response thresholds and the regulation of division of labour in insect societies. *Bull. Math. Biol.*, 60:753–807, 1998.

[9] E. Bonabeau, M. Dorigo, and G Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[10] E. Bonabeau, S. Guerín, D. Snyers, P. Kuntz, and G. Theraulaz. Three-dimensional architectures grown by simple stigmergic agents. *Biosystems*, 56 (1):13–32, 2000.

[11] M. Campos, E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg. Dynamic scheduling and division of labor in social insects. *Adaptive Behavior*, 8(2): 83–92, 2001.

[12] B. Chaib-draa. Industrial applications of distributed ai. *Communications of the ACM*, 38(11):49–53, 1995.

[13] Yann Chevaleyre, Paul E. Dunne, Ulle Endriss, Jérôme Lang, Michel Lemaître, Nicolas Maudet, Julian Padget, Steve Phelps, Juan A. Rodríguez-Aguilar, and Paulo Sousa. Issues in multiagent resource allocation.

[14] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages –1957 Vol. 3, 1999. doi: 10.1109/CEC.1999.785513.

[15] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002. doi: 10.1109/4235.985692. URL `http://dx.doi.org/10.1109/4235.985692`.

[16] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[17] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*,

94(7):1257–1270, August 2006. doi: 10.1109/JPROC.2006.876939. URL http://dx.doi.org/10.1109/JPROC.2006.876939.

[18] A. Dornhaus, F. Klügl, F. Puppe, and J. Tautz. Task selection in honeybees - experiments using multi-agent simulation. In *Proc of GWAL'98, Bochum*, 1998.

[19] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1 (1):63–83, 1989.

[20] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995. MHS '95.*, pages 39–43, 1995. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=494215.

[21] R. Eberhart, P. Simpson, and R. Dobbins. *Computational intelligence PC tools*. Academic Press Professional, Inc., San Diego, CA, USA, 1996. ISBN 0-12-228630-8.

[22] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. volume 1, pages 84–88 vol.1, 2000. doi: 10.1109/CEC.2000.870279. URL http://dx.doi.org/10.1109/CEC.2000.870279.

[23] D.B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, January 1994. doi: 10.1109/72.265956.

[24] F. Glover. Tabu search - part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[25] Fred Glover. Tabu Search–Part II. *INFORMS Journal on computing*, 2(1): 4–32, 1990.

[26] Harry Goldingay and Jort van Mourik. Genetics and competing strategies in a threshold model for mail processing. Technical Report NCRG/2008/003, Aston University, 2008.

[27] Harry Goldingay and Jort van Mourik. The influence of memory in a threshold model for distributed task assignment. In *SASO '08: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 117–126, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3404-6.

[28] P. P. Grassé. La reconstruction du nid et les interactions inter-individuelles chez les bellicositermes natalenis et cubitermes sp. la théorie de la stigmergie: essai d'interprétation des termites constructeurs. *Insectes Sociaux*, 6:41–83, 1959.

[29] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 312–317. Morgan Kaufmann, 1996.

[30] R. Hassan, B. Cohanim, and O. de Weck. A comparison of particle swarm optimization and the genetic algorithm. In *Proceedings of the 46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, 2005.

[31] J. Holland. Genetic algorithms. *Sci. Am.*, 267(1):66–72, 1992.

[32] B. Hong and V. K. Prasanna. Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. In *IPDPS*, 2004.

[33] T. Huang and A.S. Mohan. A hybrid boundary condition for robust particle swarm optimization. *Antennas and Wireless Propagation Letters, IEEE*, 4: 112–117, 2005. ISSN 1536-1225. doi: 10.1109/LAWP.2005.846166.

[34] Z. Huang and G. E. Robinson. Honeybee colony integration: Worker-worker interactions mediate hormonally regulated plasticity in division of labor. In *Proceedings of the National Academy of Sciences 89(24)*, pages 11726–11729, 1992.

[35] G. Hughes. On the mean accuracy of statistical pattern recognizers. *Information Theory, IEEE Transactions on*, 14(1):55–63, 1968. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1054102`.

[36] P. Janacik, T. Haimfarth, and F. Rammig. Emergent topology control based on division of labour in ants. In *Proceedings of the IEEE 20th International Conference on Advanced Information Networking and Applications (AINA 2006)*, 2006.

[37] R. A. Johnson. Learning, memory, and foraging efficiency in two species of desert seed-harvester ants. *Ecology*, 72(4):1408–1419, 1991.

[38] J. Kennedy and R. Eberhart. Particle swarm optimization. volume 4, pages 1942–1948, 1995. doi: 10.1109/ICNN.1995.488968. URL `http://dx.doi.org/10.1109/ICNN.1995.488968`.

[39] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*, pages 1671–1676, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7803-7282-4.

[40] O. Kittithreerapronchai and C. Anderson. Do ants paint trucks better than chickens? markets versus response thresholds for distributed dynamic scheduling. *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, 2: 1431–1439 Vol.2, Dec. 2003. doi: 10.1109/CEC.2003.1299839.

[41] Reimer Kühn, Jort van Mourik, Martin Weigt, and Annette Zippelius. Finitely coordinated models for low-temperature phases of amorphous systems. *J. Phys. A: Math. Theor.*, 40:9227–9252, 2007.

[42] K. Lerman and A. Galstyan. A general methodology for mathematical analysis of multi-agent systems. Technical Report ISI-TR-529, University of California, Information Sciences Institute, 2001.

[43] K. Lerman and A. Galstyan. Agent memory and adaptation in multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 797–803, 2003.

[44] J.J. Liang, A.K. Qin, P.N. Suganthan, and S. Baskar. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *Evolutionary Computation, IEEE Transactions on*, 10(3):281–295, June 2006. ISSN 1089-778X. doi: 10.1109/TEVC.2005.857610.

[45] Kian Hsiang Low, Wee Kheng Leow, and Marcelo H. Ang, Jr. Task allocation via self-organizing swarm coalitions in distributed mobile sensor network. In *Proc. 19th National Conference on Artificial Intelligence (AAAI-04)*, pages 28–33, 2004.

[46] Roger Mailler, Victor Lesser, and Bryan Horling. Cooperative negotiation for soft real-time distributed resource allocation. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 576–583, New York, NY, USA, 2003. ACM. ISBN 1-58113-683-8. doi: http://doi.acm.org/10.1145/860575.860667.

[47] Rui Mendes. *Population Topologies and Their Influence in Particle Swarm Performance*. PhD thesis, Universidade do Minho, 2004.

[48] Rui Mendes, James Kennedy, and JosÃĺ Neves. The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8:204–210, 2004.

[49] D. Morley. Painting trucks at general motors: The effectiveness of a complexity-based approach. *Embracing Complexity: Exploring the Application of Complex Adaptive Systems to Business*, pages 53–58, 1996.

[50] S. Nouyan, R. Ghizzioli, M. Birratari, and M. Dorigo. An insect-based algorithm for the dynamic task allocation problem. Technical Report TR/IRIDIA/205-031., UniversitÃĺ Libre de Bruxelles, 2005.

[51] E. Ozcan and C.K. Mohan. Particle swarm optimization: surfing the waves. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages –1944 Vol. 3, 1999. doi: 10.1109/CEC.1999.785510.

[52] Riccardo Poli. Analysis of the publications on the applications of particle swarm optimisation. *J. Artif. Evol. App.*, 2008(1):1–10, January 2008. ISSN 1687-6229. doi: 10.1155/2008/685175. URL `http://dx.doi.org/10.1155/2008/685175`.

[53] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, June 2007. ISSN 1935-3812 (Print) 1935-3820 (Online). doi: 10.1007/s11721-007-0002-0. URL `http://www.springerlink.com/content/046g237554721g95/`.

[54] Mitchell A. Potter and Kenneth A. De Jong. A cooperative coevolutionary approach to function optimization. In *PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, pages 249–257, London, UK, 1994. Springer-Verlag. ISBN 3-540-58484-6.

[55] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[56] R. Price. Evaluaton of adaptive nature inspired task allocation against alternate decentralised multiagent strategies, 2004.

[57] R. Price and P. Tiňo. *Parallel Problem Solving from Nature VIII*, chapter Evaluaton of Adaptive Nature Inspired Task Allocation Against Alternate Decentralised Multiagent Strategies, pages 982–990. Springer Berlin / Heidelberg, 2004.

[58] Jim Pugh, Yizhen Zhang, and Alcherio Martinoli. Particle swarm optimization for unsupervised robotic learning. In *Swarm Intelligence Symposium*, pages 92–99, 2005. URL `http://www.ieeeswarm.org`.

[59] Omer F. Rana and Kate Stout. What is scalability in multi-agent systems? In *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, pages 56–63, 2000. ISBN 1-58113-230-1.

[60] J. Riget and J.S. Vesterstrøm. A diversity-guided particle swarm optimizer - the arpso. Technical report, Aarhus Universitet, 2002.

[61] R. L. Riolo. Survival of the fittest bits. *Sci. Am.*, 267:114–116, 1992.

[62] J. Robinson and Y. Rahmat-Samii. Particle swarm optimization in electromagnetics. *IEEE Transactions on Antennas and Propagation*, 52(2):397–407, 2004.

[63] R. Schoonderwoerd, O. Holland, and J. Bruten. Ant-like agents for load balancing in telecommunications networks. In *Proceedings of the first international conference on Autonomous agents*, pages 209–216, 1997.

[64] N. N. Schraudolf and T. J. Sejnowski. Unsupervised discrimination of clustered data via optimization of binary information gain. In *Advances in Neural Information Processing Systems.*, volume 5, pages 499–506, 1993.

[65] Onn Shehory, Sarit Kraus, and Osher Yadgar. Emergent cooperative goal-satisfaction in large scale automated-agent systems. *Artificial Intelligence*, 110 (1):1–55, 1999.

[66] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73, May 1998. doi: 10.1109/ICEC.1998.699146.

[67] Andrew M. Sutton, Darrell Whitley, Monte Lunacek, and Adele Howe. Pso and multi-funnel landscapes: how cooperation might limit exploration. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 75–82, New York, NY, USA, 2006. ACM. ISBN 1-59593-186-4. doi: http://doi.acm.org/10.1145/1143997.1144008.

[68] G. Theraulaz, E. Bonabeau, and J.-L. Deneubourg. Response threshold reinforcement and division of labour in insect societies. In *Proc. Roy. Soc. London B 265*, pages 327–332, 1998.

[69] F. van den Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239, 2004. doi: 10.1109/TEVC.2004.826069. URL `http://dx.doi.org/10.1109/TEVC.2004.826069`.

[70] F. van den Bergh and A. P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937–971, April 2006. doi: 10.1016/j.ins.2005.02.003. URL `http://dx.doi.org/10.1016/j.ins.2005.02.003`.

[71] Jakob Vesterstrøm and Rene Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 IEEE Congress on Evolution-*

ary Computation, pages 1980–1987, Portland, Oregon, 20-23 June 2004. IEEE Press. ISBN 0-7803-8515-2.

[72] Darrell Whitley, Deon Garrett, and Jean paul Watson. Quad search and hybrid genetic algorithms. In *In Genetic and Evolutionary Computation - GECCO 2003*, pages 1469–1480. Springer, 2003.

A Theory

A.1 Details of the Theoretical Analysis

Exact time evolution

As discussed in section 3.3.4, the exact time evolution of the algorithm can be calculated in the large system limit when the total number of states is finite, and can be divided into four distinct phases. For the mail uptake stage, we note that agents only visit cities if their mail queue is not full, but that their behaviour at the cities is otherwise only depends on their specialisation $\boldsymbol{\theta}$. Therefore, we define the marginal densities of active agents with a given specialisation as:

$$\mu^a_{\boldsymbol{\theta}} \equiv \sum_{\boldsymbol{q}_L \ (L<L_q)} \mu_{\boldsymbol{\theta},\boldsymbol{q}_L} \tag{A.1}$$

and the total number of active agents as $N^a_a = N_a \sum_{\boldsymbol{\theta}} \mu^a_{\boldsymbol{\theta}}$.

In the current model, agents visit cities randomly such that the probability that a

subset of k agents visits any given city, is given by

$$\binom{N_a^a}{k}\left(\frac{1}{N_c}\right)^k\left(\frac{N_c-1}{N_c}\right)^{N_a^a-k} \simeq P_{R_{a/c}^a}(k), \tag{A.2}$$

where $R_{a/c}^a \equiv N_a^a/N_c$ is the ratio of active agents to cities, and P_λ is the Poisson distribution with parameter λ, which can be truncated to arbitrary precision. We also introduce short-hand notations $\langle n\rangle$ for an arbitrary subset of mail types, $\langle n\rangle_k$ ($\equiv \{n_1..n_k\}$) for a subset with k distinct mail types, and $\langle n\rangle_{k-1}^l$ the corresponding subset with mail type n_l removed.

a.1 Evolution of agent states during mail uptake

One can write down the probability $U_m(\boldsymbol{\theta}, \boldsymbol{w}, i)$ that an active agent with specialisation $\boldsymbol{\theta}$ in position $i > 1$ at a city with initial waiting times $\boldsymbol{w}$, takes mail of type m in recursive form:

$$U_m(\boldsymbol{w},\boldsymbol{\theta},i) = \sum_{\boldsymbol{\theta'}} \mu_{\boldsymbol{\theta'}}^a \left[U_0(\boldsymbol{w},\boldsymbol{\theta'})\; U_m(\boldsymbol{w},\boldsymbol{\theta},i-1) + \sum_{n\neq m} U_n(\boldsymbol{w},\boldsymbol{\theta'})\; U_m(\boldsymbol{w}_n,\boldsymbol{\theta},i-1)\right], \tag{A.3}$$

where the first term corresponds to the case that the previous agent (with thresholds $\boldsymbol{\theta'}$) did not take any mail, while the second term case corresponds to the case that the previous agent took mail type $n(\neq m)$ and the waiting times were updated to $\boldsymbol{w}_n \equiv \boldsymbol{w}|_{w_n\to 0}$. The corresponding probability that the agent takes no mail is given by $U_0(\boldsymbol{w},\boldsymbol{\theta},i) = 1 - \sum_m U_m(\boldsymbol{w},\boldsymbol{\theta},i)$.

When it is the agents' turn (i.e. when it is first in the queue, $i = 1$), the $U_m(\boldsymbol{w},\boldsymbol{\theta})$ ($\equiv U_m(\boldsymbol{w},\boldsymbol{\theta},1)$) are given by:

$$U_m(\boldsymbol{w},\boldsymbol{\theta}) = \frac{\Theta(w_m,\theta_m)}{N_m} \times \left(\sum_{k=0}^{N_m-1} \frac{1}{\binom{N_m-1}{k}} \sum_{\langle n\rangle_k(\not\ni m)} \prod_{i=1}^{k}(1-\Theta(w_{n_i},\theta_{n_i}))\right), \tag{A.4}$$

where we sum over all possible subsets of mail types that are rejected before mail type m is accepted. The corresponding probability that the agent takes no mail is given by $U_0(\boldsymbol{w},\boldsymbol{\theta}) = \prod_{m=1}^{N_m}(1-\Theta(w_m,\theta_m))$. Using these definitions, we can now easily write down the total probability $U_m(\boldsymbol{\theta})$ that an active agent with specialisation $\boldsymbol{\theta}$ takes mail of type m:

$$U_m(\boldsymbol{\theta}) = \sum_{\boldsymbol{w}} \eta_{\boldsymbol{w}} \sum_{k=1} \frac{P_{R_{a/c}^a}(k-1)}{k} \sum_{i=1}^{k} U_m(\boldsymbol{w},\boldsymbol{\theta},i), \tag{A.5}$$

where we sum over all possible sets of waiting times at the cities, numbers of visiting agents and positions of the agent in that set. Note that an active agent always visits a city, such that the sum starts at $k = 1$, and that $P_{R^a_{a/c}}(k-1)$ is the probability that $k-1$ *other* agents visit that particular city. Again, $U_0(\boldsymbol{\theta}) = 1 - \sum_m U_m(\boldsymbol{\theta})$ is the probability that it takes no mail. Note, furthermore, that the inactive agents all have $U_0(\boldsymbol{\theta}) = 1$ irrespective of their specialisation. Then, the element of the transition matrix $\mathbf{T}$ describing (the probability of) a transition from an agent state $(\boldsymbol{\theta}_m, \boldsymbol{q}_K)$ to a state $(\boldsymbol{\theta}_n, \boldsymbol{q}'_L)$ during the mail uptake stage is given by:

$$T_{(\boldsymbol{\theta}_n,\boldsymbol{q}'_L),(\boldsymbol{\theta}_m,\boldsymbol{q}_K)} = U_0(\boldsymbol{\theta}_n)\ \delta_{n,m}\ \delta_{\boldsymbol{q}'_L,\boldsymbol{q}_K} + \delta_{L,K+1} \prod_i^K \delta_{q'_i,q_i} \times \left(U_n(\boldsymbol{\theta}_n)\ \delta_{n,m}\ \delta_{q'_L,t_p} + U_n(\boldsymbol{\theta}_m)\ (1-\delta_{n,m})\ \delta_{q'_L,t_c} \right) \tag{A.6}$$

where $\delta_{a,b}$ is the Kronecker delta (i.e. 1 if its arguments are equal, and 0 otherwise), while $\boldsymbol{\theta}_n$ is the set of thresholds for specialisation n (i.e. $\theta_i = \theta_{min}$ if $i = n$, and $\theta_i = \theta_{max}$ otherwise). Note that the state either remains unchanged (when no mail was taken), or a processing time t_p is added to the queue (when mail of the current specialisation was taken), or a change-over time t_c is added to the queue and the specialisation is changed (when mail of a different type was taken).

The efficiency can also be expressed in terms of the probabilities of mail uptake and is given by:

$$E = \sum_{\boldsymbol{\theta}} \mu^a_{\boldsymbol{\theta}} \sum_{m=1}^{N_m} U_m(\boldsymbol{\theta}) \tag{A.7}$$

c.1 Evolution of city states during mail uptake

Similarly, we can write down the probability $G_{\langle n\rangle_k}(\boldsymbol{w}, i)$ that exactly the subset $\langle n\rangle_k$ of mail types is given out by a city with waiting times $\boldsymbol{w}$ and i visiting agents, in recursive form:

$$G_{\langle n\rangle_k}(\boldsymbol{w}, i) = \sum_{\boldsymbol{\theta}} \mu^a_{\boldsymbol{\theta}} \left(\sum_{l=1}^{k} U_{n_l}(\boldsymbol{w}, \boldsymbol{\theta})\ G_{\langle n\rangle^l_{k-1}}(\boldsymbol{w}_{n_l}, i-1) + U_0(\boldsymbol{w}, \boldsymbol{\theta})\ G_{\langle n\rangle_k}(\boldsymbol{w}, i-1) \right) \tag{A.8}$$

with $G_{\langle n\rangle_k}(\boldsymbol{w}, i) = 0$, when $k > i$ or $w_m = 0$ for any $m \in \langle n\rangle_k$, and with $G_{\langle n\rangle_0}(\boldsymbol{w}, 0) = 1$. The total probability $G_{\langle n\rangle}(\boldsymbol{w})$ that exactly a subset $\langle n\rangle$ of mail types is given out by a city with waiting times $\boldsymbol{w}$ is given by:

$$G_{\langle n\rangle}(\boldsymbol{w}) = \sum_{i=0} P_{R^a_{a/c}}(i)\ G_{\langle n\rangle}(\boldsymbol{w}, i) \tag{A.9}$$

Then, the element of transition matrix $\mathbf{L}$ describing (the probability of) a transition from an city state $\boldsymbol{w}$ to a state $\boldsymbol{w}'$ during the mail uptake stage is given by:

$$L_{\boldsymbol{w}',\boldsymbol{w}} = G_{\langle n_{\boldsymbol{w}-\boldsymbol{w}'}\rangle}(\boldsymbol{w}) \tag{A.10}$$

where $\langle n_{\boldsymbol{w}-\boldsymbol{w}'}\rangle$ is the set of indices n for which $w_n \neq 0$ and $w'_n = 0$.

a.2 Evolution of agent states during queue processing

The element of the transition matrix $\mathbf{Q}$ describing (the probability of) a transition from an agent state $(\boldsymbol{\theta}, \boldsymbol{q}_K)$ to a state $(\boldsymbol{\theta}', \boldsymbol{q}'_L)$ during the queue processing stage is relatively straightforward to write down directly:

$$\begin{aligned} Q_{(\boldsymbol{\theta}',\boldsymbol{q}'_L),(\boldsymbol{\theta},\boldsymbol{q}_K)} \;=\; \delta_{\boldsymbol{\theta},\boldsymbol{\theta}'} \times \Bigg(& \delta_{K,0}\delta_{L,0} + \\ & (1-\delta_{q_1,1})\delta_{L,K}\delta_{q'_1,q_1-1}\prod_{i=2}^{K}\delta_{q'_i,q_i} + \delta_{q_1,1}\delta_{L,K-1}\prod_{i=1}^{K}\delta_{q'_i,q_{i+1}} \Bigg). \end{aligned} \tag{A.11}$$

Note that specialisations don't change, that empty queues remain empty, and that the first element of non empty queues gets lowered by one until it reaches 0, in which case all the other elements in the queue are shifted by one position.

c.2 Evolution of city states during mail production

The element of transition matrix $\mathbf{P}$ describing (the probability of) a transition from an city state $\boldsymbol{w}$ to a state $\boldsymbol{w}'$ during the mail production stage is again relatively straightforward to write down directly:

$$\begin{aligned} P_{\boldsymbol{w}',\boldsymbol{w}} = \prod_{m=0}^{N_m} \Big(& (1-\delta_{w_m,0})\ \delta_{w'_m,\min(w_m+1,\theta_{max}+1)} + \\ & \delta_{w_m,0}\ (\pi_m(t)\ \delta_{w'_m,1} + (1-\pi_m(t))\ \delta_{w'_m,0}\ ) \Big), \end{aligned} \tag{A.12}$$

where the $\pi_m(t)$ are time dependent for the dynamic environment only.

A.2 Theoretical Upper Bound for the Efficiency

We derive the upper bound for the efficiency in *ideal circumstances*, when no mail is lost due to $\boldsymbol{\ell}$.1-$\boldsymbol{\ell}$.3, and the efficiency is only limited by $\boldsymbol{\ell}$.4. Then all agents have an identical set of thresholds $\boldsymbol{\theta} = \{\theta_n(=\theta_{min}=0), \forall n = 1..N_m\}$. Mail uptake

is independent of the waiting time, and only depends on the availability of the different mail types. Hence, the city states can be simplified to $\mathcal{C} = \boldsymbol{b} \in \{0,1\}^{N_m}$ where $b_m = 1$ when the mail type m is available, and $b_m = 0$ when it is not.

Agents visit cities randomly such that the probability that a subset of i agents visits any given city, is given by equation (A.2) Since agents are indistinguishable, the probability that an agent takes mail when visiting a city with k available mail types and i visiting agents, is given by:

$$U(k,i) = \begin{cases} \frac{k}{i} \ , & \text{if } i > k \ , \\ 1 \ , & \text{if } i \leq k \ . \end{cases} \tag{A.13}$$

The total probability that an agent takes mail (i.e. the efficiency) is thus:

$$\begin{aligned} E(t) \ &= \ \sum_{k=1}^{N_m} \chi_k(t) \ \sum_j P_{R_{a/c}}(j-1) \ U(k,j) = \\ &\sum_{k=1}^{N_m} \chi_k(t) \left(1 - P_{R_{a/c}}(k) + \frac{k - R_{a/c}}{R_{a/c}} (1 - \sum_{j=0}^{k} P_{R_{a/c}}(j)) \right) . \end{aligned} \tag{A.14}$$

where $\chi_k(t) \equiv \sum_{\boldsymbol{b} \in \mathcal{S}_\mathcal{C}} \chi_{\boldsymbol{b}}(t) \delta_{|\boldsymbol{b}|,k}$ is the probability that a city has exactly k pieces of mail available.

We now derive the expressions of the matrices $\mathbf{P}(t)$ and $\mathbf{L}$, that govern exact time evolution for the city profile:

$$\boldsymbol{\chi}(t+1) = \mathbf{P}(t) \ \mathbf{L} \ \boldsymbol{\chi}(t) \ . \tag{A.15}$$

1. Evolution of city states during mail uptake

We can now write down the elements of the transition matrix $\mathbf{L}$ describing (the probability of) a transition from an city state $\boldsymbol{b}$ to a state $\boldsymbol{b}'$ during the mail uptake stage. It is clear that

$$L_{\boldsymbol{b}',\boldsymbol{b}} = \sum_{i=0}^{\infty} P_{R_{a/c}}(i) \ L_{\boldsymbol{b}',\boldsymbol{b}}(i) \tag{A.16}$$

where $L_{\boldsymbol{b}',\boldsymbol{b}}(i)$ is the corresponding transition probability for a city visited by exactly i agents. Since all agents take mail when available, and since during the uptake phase all mail types are equivalent, we have that

$$L_{\boldsymbol{b}',\boldsymbol{b}}(i) = \begin{cases} \delta_{\boldsymbol{b}',\boldsymbol{0}}, & i \geq |\boldsymbol{b}| \\ \binom{|\boldsymbol{b}|}{i}^{-1} \ \delta_{|\boldsymbol{b}'|,|\boldsymbol{b}|-i} \ \prod_{m=1}^{N_m} \left(1 - \delta_{b'_m,1} \delta_{b_m,0}\right) , & i < |\boldsymbol{b}| \end{cases} \tag{A.17}$$

In the first case, there are enough agents and all mail is taken. In the second case a random subset of i out of $|\boldsymbol{b}|$ mail types is taken ($\binom{|\boldsymbol{b}|}{i}$ possibilities), such that there are $|\boldsymbol{b}| - i$ mail types left, and these must be of the types that were originally present.

2. Evolution of city states during mail production

The element of transition matrix $\mathbf{P}$ describing (the probability of) a transition from an city state $\boldsymbol{b}$ to a state $\boldsymbol{b}'$ during the mail production stage is relatively straightforward to write down directly:

$$P_{\boldsymbol{b}',\boldsymbol{b}} = \prod_{m=1}^{N_m} \left(\delta_{b_m,1}\ \delta_{b'_m,1} + \delta_{b_m,0}\ (\pi_m(t)\ \delta_{b'_m,1} + (1-\pi_m(t))\ \delta_{b'_m,0}) \right), \tag{A.18}$$

where the $\pi_m(t)$ are time dependent for the dynamic environment only.

B Parameter Details

B.1 The Standard Setting

The following parameters define the **standard setting** for the mail processing problem. We set $N_a = 5 \cdot 10^4$ and, in order to have a fair comparison between different environments, we take and $R_{a/m} = 1$ in a static environment, while in the dynamic environment we take $R_{a/m} = 0.5$ (as $\overline{\pi_m(t)} = 0.5$ over a period) and we fix $N_m = 2$. The standard dynamic environment has a period $\xi = 50$ and we fix $t_p = 1$ and $t_c = 10$. A standard run consists of 500 iterations over which the average efficiency per agent is monitored.

For reasons discussed in the main text we also define a **standard MB setting** to allow meaningful comparisons between threshold and market based algorithms. This is identical to the standard setting with the exception of setting $t_c = 2$, the reasons for which are discussed in the main text.

For ease of reference, we reproduce the parameters for our algorithms in tables B.1, B.2 & B.3 below.

Table B.1: Parameters for threshold based algorithms

Parameter	θ_{min}	θ_{max}	ϵ, ψ	α, β, η	λ
Value	0	50	5	0.5	2

Table B.2: Parameters for market based algorithms

Parameter	ω_s	ω_t
Value	1750	4

Table B.3: Parameters for memory algorithms

Parameter	μ_a	$\mathcal{L}_a$	ρ
Value	10	10	0.95

B.2 Parameter Optimisation

While fully investigating the effect of the parameters on our algorithm is beyond the scope of this paper, we do wish to be able to compare the performance of our algorithms in different circumstances. As such, we employ a robust PSO algorithm in order to find parameters which give us good algorithm performance. Note that we do not claim that these parameters are necessarily the truly optimal set, merely that they allow each algorithm performing well, unhampered by our initial parameter choices.

Each run of the PSO algorithm consists of 20 particles and is stopped after $5 \cdot 10^4$ function evaluations. Here a single function evaluation consists of a single run of the task allocation algorithm we are attempting to optimise, in the environment for which we are trying to optimise it, and the output value is the average efficiency over the course of this run. Each run consists of 500 iterations with $1 \cdot 10^2$ agents, although the final results presented are for the parameters tested on $5 \cdot 10^4$ agents. Other than the number of agents used, the only deviation from the standard setting is, when specified, using a value of $t_c = 2$. Note although PSO is generally framed as a minimisation algorithm, we can minimise the efficiency multiplied by -1 which is

exactly equaivlent to maximising efficiency. The initial parameters used in the PSO algorithm are set using the method defined by equation 2.10, giving $\eta(0) \approx 0.7298$ and $c_1 = c_2 \approx 1.496$. We then linearly decrease inertial weight with the number of function evaluations, so that

$$\eta(t) = \eta(0)\left(1 - \frac{f_n(t)}{f_{max}}\right) \tag{B.1}$$

where $f_n(t)$ is the number of function evaluations at time t and $f_{max} = 5 \cdot 10^4$ is the maximum number of function evaluations.

We choose to optimise three different algorithms. Fistly, we will optimise the SO rule and the market based algorithm as they outperformed the VRT rule in all circumstances. We also choose to optimise the hybrid VRT algorithm. Not only does this allow us to test whether our hybrid algorithm can outperform the individual algorithms upon which it is based but, if it does not, its ability to evolve into any of our ther algorithms (including the VRT rule) gives us a method to find which single algorithm perfoems best.

As an exhaustive investigation of the optimised performance of these algorithms under all possible combinations of system parameters (or even under individual variation of each system parameter) would be impractical, we choose to test the algorithms' performances in four representative sttings: the static and dynamic settings with $t_c = 10$, and the static and dynamic settings with $t_c = 2$. Performance in the static and dynamic settings can be seen respectively as representative of an algorithms' performance when stability, or when re-specialisation ability is required. Performance when $t_c = 10$ can be seen as representative of an algorithm's performance when it is overworked and when selectivity is required so as not to cause a fatally compromising cascade of agent faliures. If $t_c = 2$, we can see this as representating a situation in which tasks are sparse and, therefore, in which immediate task completion is more important than the short-term performance of individual agents.

B.2.1 Re-parametrisation

Before proceeding with the optimisation of our algorithms it is useful to determine whether their "natural" parameter set, as given in their definitions, is conducive to

optimisation. In order to do this, we will consider what problems might arise when optimising these natural parameters.

Firstly, we require $\theta_{max} \geq \theta_{min}$ but as θ_{min} in a parameter in our PSO algorithm the bounding mechanism required to achieve this directly would be complex and could compromise our algorithm's performance. Therefore, instead of optimising θ_{max} directly, we choose to optimise a parameter p_1 such that $\theta_{min} + p_1 = \theta_{max}$ and ensure $p_1 \geq 0$. Secondly, it is clear that the re-specialisation constants for the VRT rule, ϵ and ψ, should be restricted to the range $[0, \theta_{max} - \theta_{min}]$. For similar reasons as θ_{max} this is impractical. Therefore, we define parameters, p_2 and p_3, where $\epsilon = p_2(\theta_{max} - \theta_{min})$ and likewise $\psi = p_3(\theta_{max} - \theta_{min})$ and optimise them. Restricting these parameters to the interval $[0, 1]$ will restrict ϵ and ψ to the desired range. It is also possible that this re-parametrisation will lead to better search performance as p_1 and p_2 are a direct measure of how quickly an agent can re-specialise.

Finally, we have two parameters, λ in the ETF and ω_t in the market bidding function which are used as exponents. This means that a small change in either of these parameters can greatly affect the overall performance of the algorithm. This is particularly true for small values of either parameter because for high values of either parameter the functions effectively become discrete. In the case of the ETF, a high value of λ gives us a function of the form

$$\Theta(s, \theta) \approx \begin{cases} 0 & \text{if } w < \theta \\ 0.5 & \text{if } w = th \\ 1 & \text{otherwise} \end{cases} \tag{B.2}$$

whereas for high ω_t we have a bidding function with approximately the following properties

$$B_a(m) > B_b(m) \begin{cases} \text{if} & T_a(m) < T_b(m) \\ \text{else if} & T_a(m) = T_b(m) \quad \text{and} \quad \delta_{m,\sigma_a} > \delta_{m,\sigma_b} \end{cases} \tag{B.3}$$

As, for both of these parameters, we have a situation in which high values lead to similar behaviour but a minor change in a small value may lead to a large behavioural change, we wish our algorithm to have fine control over small parameter values. Therefore instead of working directly with λ and ω_t, we will instead work with new parameters p_4 and p_5, where $\lambda = e^{p_4}$ and $\omega_t = e^{p_5}$.

The ranges we permitted these parameters to take are given in table B.4.

Table B.4: Parameter bounds for the PSO algorithm

Parameter	Minimum Value	Maximum Value
θ_{min}	0	100
$p_1 = \theta_{max} - \theta_{min}$	0	100
$p_2 = \frac{\epsilon}{\theta_{max} - \theta_{min}}$	0	1
$p_3 = \frac{\psi}{\theta_{max} - \theta_{min}}$	0	1
$p_4 = \ln \lambda$	$\ln 1$	$\ln 10$
ω_s	0	2000
$p_5 = \ln \omega_t$	$\ln 10^{-10}$	$\ln 10$

Table B.5: Optimised parameters for the market based algorithm

Environment	Static		Dynamic	
t_c	10	2	10	2
ω_s	1948.02	764.79	1009.03	108.261
ω_t	$5.0941 \cdot 10^{-5}$	$6.4002 \cdot 10^{-2}$	$4.6837 \cdot 10^{-5}$	1.223
Original Efficiency	0.246390	0.729365	0.250129	0.653368
PSO Efficiency	0.246399	0.729369	0.249985	0.653368

B.2.2 Parameter Details

We will now discuss the results obtained from our PSO algorithm runs. It is important to note that as these represent the output of single runs of an optimisation algorithm, rather than a systematic investigation of the parameters, it is inappropriate to draw strong conclusions from the parameter values obtained. However, we will try to highlight the patterns within the parameter values and suggest reasons for them.

While optimising the market based algorithm, it is interesting to note that for each setting efficiencies very close to the final optimised values were found within a

Table B.6: Optimised parameters for the SO rule

Environment	Static		Dynamic	
t_c	10	2	10	2
θ_{min}	0	0	0	0
θ_{max}	4.4746	0.43421	17.8449	0.0152564
λ	7.2969	1.0266	1.0062	7.2174
Original Efficiency	0.579069	0.579163	0.463695	0.463681
PSO Efficiency	0.630432	0.704393	0.513024	0.653255

single iteration (20 function evaluations) of PSO. This indicates that there may be a wide range of parameters for which the market based algorithm performs well. This hypothesis is supported by the efficiencies shown in table B.5, which shows that the parameters found by the PSO algorithm perform almost identicaly to those we used originally. Although this is not entirely surprising, considering that our original market parameters were based on the output of a GA, the fact that such different parameters can produce such similar results indicates that we have a reasonably wide margin for error in our parameter choices. It is also clear that the lack of selectivity is a inherent flaw in the market based alogrithm for high t_c as no parameter set was able to overcome the high level of $\boldsymbol{\ell}.1$ and produce reasonable performance.

Optimising the SO rule leads to a different outcome. While good parameter sets were found within the first few PSO iterations, the initial best positions had very low efficiency. This low efficiency appears to be mainly due to non-zero θ_{min}, which allows $\boldsymbol{\ell}.2$ to occur and also increases $\boldsymbol{\ell}.3$. Following parameter sets with $\theta_{min} = 0$ being discovered, the rate of improvement dropped, but was still greater than was shown during the optimisation of the market based algorithm. While this suggests a lower range of parameters produce acceptable performance in the SO rule, a reasonably disperate set of good parameters were found.

The outcome of this optimisation is shown in table B.6. We can see that the optimised parameter set show a large improvement over our original arbitrarily chosen parameter set. It is interesting to note the lower values of θ_{max} when $t_c = 2$

Table B.7: Optimised parameters for the hybrid VRT algorithm

Environment	Static		Dynamic	
t_c	10	2	10	2
θ_{min}	0	0	0	0
θ_{max}	8.5015	$4.8841 \cdot 10^{-5}$	12.6245	0
ϵ	8.4993	$2.8772 \cdot 10^{-5}$	12.6188	0
ψ	8.2894	$2.3320 \cdot 10^{-5}$	12.3903	0
λ	1.1894	7.8457	1.0004	1
ω_s	1795.76	1981.15	0	658.981
ω_t	$2.6414 \cdot 10^{-6}$	$3.0478 \cdot 10^{-1}$	$1.6472 \cdot 10^{-3}$	3.4410
Efficiency	0.645127	0.72926	0.536984	0.653203

compared to in the standard setting, implying that the low cost of changeovers can allow agents to be less selective. In particular, the low value of θ_{max} when coupled with the high value of λ the dynamic environment when $t_c = 2$ means that the agents will take all mail that they encounter. They can afford to do this as the mail types they are likely to encounter in successive iterations are inherently correlated in the dynamic environment, reducing the probability of repeated changeovers in a short space of time.

In terms of performance, the optimisation of the hybrid VRT algorithm proceeded in a similar manner as the optimisation of the SO rule, with a sharp initial increase followed by a series of small improvements. This again is due mainly to the great suboptimality of non-zero θ_{min}. We can see a clear split in the parameters between those optimal inthe standard environment and those optimal when $t_c = 2$. In the standard environment the hybrid algorithm is primarily threshold based, as the high θ_{max} means that agents will tend not accept the first piece of mail offered to them if it is not of their specialised type. We also note that, rather that a VRT rule threshold algorithm, the hybrid tends to a SO style update rule. When $t_c = 2$, however, the hybrid tends to an entirely market based approach as, with $\theta_{max} \approx 0$,

agents will accept all batches of mail offered to them and so the only important detail of the algorithm is the bidding function.

C PSO Cost Results

In this appendix we give the details of the average cost of our algorithms, corresponding to the robustness results in section 5.3.2. Given in the tables are the average cost of the best solution an algorithm has found after a set number of function evaluations ($1 \cdot 10^6$ unless specified) $\pm$ 1 standard deviation.

As these results are largely explained by robustness, we will not examine them in great detail. It is, however worth noting that DiPSO's ability to re-converge to a solution, effectively repeatedly performing a local search, without the limitation of a strict minimum diversity threshold allows it to have the lowest average cost more often than would be solely explained by robustness.

Table C.1: Average Cost of the PSO algorithms in 10 dimensions

Function	Base	No Convergence	ARPSO	DiPSO
Rosenbrock	Standard	0.578 ± 1.409	0.347 ± 1.128	$\mathbf{6.875 \cdot 10^{-6} \pm 6.265 \cdot 10^{-6}}$
	CLPSO	0.0903 ± 0.532	0.0903 ± 0.532	0.003946 ± 0.02136
Sphere	Standard	$\mathbf{7.826 \cdot 10^{-9} \pm 4.533 \cdot 10^{-9}}$	$8.203 \cdot 10^{-9} \pm 4.422 \cdot 10^{-9}$	$9.178 \cdot 10^{-9} \pm 4.596 \cdot 10^{-9}$
	CLPSO	$9.512 \cdot 10^{-9} \pm 4.582 \cdot 10^{-9}$	$9.438 \cdot 10^{-9} \pm 4.646 \cdot 10^{-9}$	$1.044 \cdot 10^{-8} \pm 4.851 \cdot 10^{-9}$
Ackley	Standard	0.006267 ± 0.0896	$1.590 \cdot 10^{-7} \pm 6.226 \cdot 10^{-8}$	$\mathbf{1.224 \cdot 10^{-8} \pm 4.878 \cdot 10^{-9}}$
	CLPSO	$1.527 \cdot 10^{-8} \pm 5.342 \cdot 10^{-9}$	$8.968 \cdot 10^{-6} \pm 2.444 \cdot 10^{-6}$	$3.415 \cdot 10^{-8} \pm 1.135 \cdot 10^{-8}$
Griewank	Standard	0.0504 ± 0.0302	0.0401 ± 0.0249	0.0349 ± 0.0218
	CLPSO	$3.699 \cdot 10^{-5} \pm 0.000523$	$\mathbf{1.726 \cdot 10^{-5} \pm 0.0003900}$	$2.960 \cdot 10^{-5} \pm 0.0004681$
Rastrigin	Standard	4.736 ± 2.479	0.461 ± 1.610	$\mathbf{8.556 \cdot 10^{-9} \pm 4.500 \cdot 10^{-9}}$
	CLPSO	0.0009950 ± 0.0315	$1.202 \cdot 10^{-8} \pm 5.109 \cdot 10^{-9}$	$9.932 \cdot 10^{-9} \pm 4.905 \cdot 10^{-9}$
Rana	Standard	79.627 ± 25.724	70.882 ± 32.553	17.625 ± 13.699
	CLPSO	5.659 ± 3.241	5.659 ± 3.241	$\mathbf{3.604 \pm 2.822}$
Schwefel	Standard	265.568 ± 170.022	11.917 ± 70.042	0.477 ± 10.672
	CLPSO	2.2505 ± 17.059	$\mathbf{0.0001273 \pm 5.584 \cdot 10^{-9}}$	$\mathbf{0.0001273 \pm 4.642 \cdot 10^{-9}}$

Table C.2: Average Cost of the PSO algorithms in 30 dimensions

Function	Base	No Convergence	ARPSO	DiPSO
Rosenbrock	Standard	1.010 ± 1.736	0.724 ± 1.539	$\mathbf{0.103 \pm 0.612}$
	CLPSO	0.367 ± 2.495	0.367 ± 2.495	0.12718 ± 1.06733
Sphere	Standard	$\mathbf{1.224 \cdot 10^{-8} \pm 4.847 \cdot 10^{-9}}$	$1.401 \cdot 10^{-8} \pm 4.892 \cdot 10^{-9}$	$1.531 \cdot 10^{-8} \pm 5.337 \cdot 10^{-9}$
	CLPSO	$1.805 \cdot 10^{-8} \pm 5.329 \cdot 10^{-9}$	$2.209 \cdot 10^{-8} \pm 5.555 \cdot 10^{-9}$	$2.350 \cdot 10^{-8} \pm 6.19685 \cdot 10^{-9}$
Ackley	Standard	0.355 ± 0.686	$3.687 \cdot 10^{-6} \pm 8.492 \cdot 10^{-7}$	$\mathbf{2.820 \cdot 10^{-8} \pm 6.984 \cdot 10^{-9}}$
	CLPSO	$3.906 \cdot 10^{-8} \pm 6.692 \cdot 10^{-9}$	$7.646 \cdot 10^{-5} \pm 1.566 \cdot 10^{-5}$	$1.856 \cdot 10^{-7} \pm 1.09361 \cdot 10^{-7}$
Griewank	Standard	0.009987 ± 0.0149	0.008657 ± 0.0117	0.009784 ± 0.0134
	CLPSO	$\mathbf{1.893 \cdot 10^{-8} \pm 5.415 \cdot 10^{-9}}$	$7.416 \cdot 10^{-6} \pm 0.0002341$	$1.484 \cdot 10^{-5} \pm 0.000332053$
Rastrigin	Standard	55.874 ± 14.627	16.281 ± 27.744	$\mathbf{1.460 \cdot 10^{-8} \pm 5.135 \cdot 10^{-9}}$
	CLPSO	0.0279 ± 0.165	0.0122 ± 0.109	$2.377 \cdot 10^{-8} \pm 6.20596 \cdot 10^{-9}$
Rana	Standard	165.348 ± 22.700	158.848 ± 31.231	77.642 ± 18.877
	CLPSO	28.537 ± 5.791	28.537 ± 5.791	$\mathbf{18.558 \pm 5.538}$
Schwefel	Standard	2731.8 ± 634.78	1162.88 ± 1353.28	90.138 ± 154.861
	CLPSO	17.056 ± 43.334	9.244 ± 33.131	$\mathbf{0.0003819 \pm 6.045 \cdot 10^{-9}}$

Table C.3: Average Cost of the PSO algorithms in 100 dimensions

Function	Base	Standard	ARPSO	DIPSO
Rosenbrock	Standard	117.009 ± 48.735	$\mathbf{95.109 \pm 45.914}$	114.649 ± 47.986
	CLPSO	155.66 ± 55.603	158.853 ± 54.443	157.773 ± 57.473
Sphere	Standard	$1.308 \cdot 10^{-6} \pm 1.552 \cdot 10^{-5}$	$4.145 \cdot 10^{-8} \pm 6.683 \cdot 10^{-9}$	$\mathbf{3.740 \cdot 10^{-8} \pm 8.181 \cdot 10^{-9}}$
	CLPSO	$5.441 \cdot 10^{-8} \pm 8.426 \cdot 10^{-9}$	$7.898 \cdot 10^{-8} \pm 9.897 \cdot 10^{-9}$	$1.004 \cdot 10^{-7} \pm 1.837 \cdot 10^{-8}$
Ackley	Standard	6.809 ± 2.886	$6.608 \cdot 10^{-5} \pm 1.051 \cdot 10^{-5}$	1.495 ± 0.864
	CLPSO	0.166 ± 0.363	$3.489 \cdot 10^{-4} \pm 2.816 \cdot 10^{-5}$	$\mathbf{1.282 \cdot 10^{-6} \pm 7.933 \cdot 10^{-7}}$
Griewank	Standard	0.186 ± 0.829	0.0516 ± 0.0774	$1.787 \cdot 10^{-4} \pm 1.282 \cdot 10^{-3}$
	CLPSO	$5.181 \cdot 10^{-5} \pm 7.042 \cdot 10^{-4}$	$\mathbf{4.454 \cdot 10^{-5} \pm 5.749}$	$6.417 \cdot 10^{-5} \pm 8.456 \cdot 10^{-4}$
Rastrigin	Standard	373.288 ± 55.312	203.649 ± 100.385	82.955 ± 27.284
	CLPSO	3.915 ± 2.135	$\mathbf{3.896 \pm 2.094}$	4.793 ± 2.515
Rana	Standard	201.656 ± 16.854	199.067 ± 20.871	165.82 ± 15.061
	CLPSO	117.291 ± 10.404	117.291 ± 10.404	$\mathbf{81.288 \pm 18.432}$
Schwefel	Standard	13447.4 ± 1437.42	8317.3 ± 4511.38	7474.74 ± 1285.81
	CLPSO	283.799 ± 181.662	273.049 ± 179.389	$\mathbf{22.839 \pm 53.415}$

Table C.4: Average Cost of the PSO algorithms in 100 dimensions after 10^7 function evaluations

Function	Base	Standard	ARPSO	DIPSO
Rosenbrock	Standard	15.814 ± 25.902	$\mathbf{4.558 \pm 11.417}$	15.180 ± 26.196
	CLPSO	114.599 ± 58.196	116.656 ± 56.122	115.105 ± 57.751
Sphere	Standard	$1.823 \cdot 10^{-5} \pm 5.550 \cdot 10^{-4}$	$\mathbf{3.304 \cdot 10^{-8} \pm 6.127 \cdot 10^{-9}}$	$3.692 \cdot 10^{-8} \pm 8.648 \cdot 10^{-9}$
	CLPSO	$4.593 \cdot 10^{-8} \pm 6.989 \cdot 10^{-9}$	$6.474 \cdot 10^{-8} \pm 8.349 \cdot 10^{-9}$	$3.522 \cdot 10^{-8} \pm 6.119 \cdot 10^{-9}$
Ackley	Standard	5.993 ± 2.822	$2.004 \cdot 10^{-5} \pm 2.404 \cdot 10^{-6}$	$3.644 \cdot 10^{-4} \pm 0.0110$
	CLPSO	0.0432 ± 0.196	$0.0002232 \pm 1.446 \cdot 10^{-5}$	$\mathbf{1.237 \cdot 10^{-7} \pm 1.480 \cdot 10^{-8}}$
Griewank	Standard	0.150 ± 0.606	0.03852 ± 0.0612	0.0386 ± 0.0571
	CLPSO	$1.730 \cdot 10^{-5} \pm 3.898 \cdot 10^{-4}$	$2.105 \cdot 10^{-4} \pm 5.740 \cdot 10^{-3}$	$\mathbf{7.437 \cdot 10^{-6} \pm 2.345 \cdot 10^{-4}}$
Rastrigin	Standard	369.676 ± 54.322	85.511 ± 133.553	1.369 ± 3.261
	CLPSO	2.667 ± 1.672	2.038 ± 1.447	$\mathbf{1.990 \cdot 10^{-3} \pm 0.0448}$
Rana	Standard	198.181 ± 16.652	186.667 ± 33.549	164.964 ± 16.753
	CLPSO	84.358 ± 8.114	84.358 ± 8.114	$\mathbf{51.811 \pm 7.049}$
Schwefel	Standard	13225.2 ± 1427.09	6875.75 ± 4983.11	5678.11 ± 1514.11
	CLPSO	255.69 ± 175.348	235.002 ± 171.003	$\mathbf{0.120 \pm 3.749}$