Particle swarm optimisation:
An algorithm using support vector classification based constraint approximations

by

Maria Magdalena Malan

Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Mechanical) in the Faculty of Engineering at Stellenbosch University

Supervisor: Prof. G. Venter

April 2019

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: April 2019

Abstract

Particle swarm optimisation:
An algorithm using
support vector classification based
constraint approximations

M. M. Malan

Department of Mechanical and Mechatronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MEng Research (Mechanical)

April 2019

Particle swarm optimisation (PSO) is by its nature an unconstrained global optimisation method, which must be, and has been, adapted in order to be capable of solving constrained optimisation problems. PSO, along with other similar metaheuristics, struggles when the global optimum is located on the boundary of the feasible region, which is common in most real-world problems, and when there are equality constraints.

The aim was to develop a method for improving PSO's ability to solve these types of constrained optimisation problems. The view that was taken was that the problem with finding global optima on the boundary is the lack of knowledge about the boundary and that deliberately encouraging the exploration of the boundary is critical for having the swarm discover these optima or having the swarm discover them in fewer iterations.

To address the lack of knowledge of the boundary, support vector classification (SVC) and the data points that are already evaluated by the swarm were used to create models of the feasibility boundary. The new knowledge that these SVC models provide is used to encourage the particles' exploration of the boundary, in order to increase the likelihood of locating the global optimum.

A thorough literature review is presented to place the concepts into their proper context, present related or similar work and provide the necessary background knowledge.

The reasoning behind the concepts that were created and their implementation is laid out. Four concepts with several variations were designed and

evaluated through testing on a large set of test problems. The concepts were considered to be additions to a baseline PSO algorithm, and the impact their addition had was evaluated relative to it. The concepts were assessed while mimicking the SVC's predictions, thus assuming 100% model accuracy, and then with the full classifier attached.

Overall, several of the concepts provided significant reductions in the number of iterations that were required on many of the test problems. There were also clear improvements on some of the simpler equality constrained problems. Some problems or challenges are explained, and suggestions are made for improving any future implementations.

The most generally promising concept algorithm shifts particles for which the position rule's new position would entail a move from feasible to infeasible region, back to the approximate point the particle would have to cross the feasibility boundary.

The intended application is to optimisation problems where the evaluation of the objective function and constraints significantly dominates the computation time, as in larger engineering design problems with simulations. For these problems the computational overhead that is introduced by the creation and automated tuning of the SVC models could potentially be negligible relative to the overall decrease associated with the reduction in the number of function evaluations required.

Uittreksel

Deeltjie swerm optimering:
'n Algoritme wat
ondersteuningsvektor klassifikasie gebaseerde
beperking benaderings gebruik

("Particle swarm optimisation:
An algorithm using support vector classification based constraint approximations")

M. M. Malan

Departement van Meganiese en Megatroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid-Afrika.

Tesis: MIng Navorsing (Meganies)

April 2019

Deeltjie swerm optimering (DSO) is volgens sy aard 'n onbeperkte globale optimeringsmetode wat aangepas moet word ten einde beperkte optimeringsprobleme te kan oplos. DSO, saam met ander soortgelyke metaheuristieke, sukkel wanneer die globale optimum op die grens van die toelaatbare streek geleë is, wat algemeen gebeur in regte-wêreld probleme, en asook wanneer daar gelykheidsbeperkings is.

Die doel was om 'n metode te ontwikkel om DSO se vermoë om hierdie tipe beperkte optimeringsprobleme op te los, te verbeter. Die siening was dat die probleem met die vind van globale optima op die grens die gebrek aan kennis oor die grens is en dat die verkenning van die grens gebied doelbewus aangemoedig moet word, ten einde die swerm in staat te stel om hierdie optima te ontdek of dit in minder iterasies te ontdek.

Om die gebrek aan kennis van die grens aan te spreek, word ondersteuningsvektor klassifikasie (OVK) en die data punte wat reeds deur die swerm geëvalueer is gebruik om modelle van die toelaatbaarheidsgrens te skep. Die nuwe kennis wat hierdie OVK-modelle bied, word aangewend om die deeltjies se verkenning van die grens aan te moedig ten einde die waarskynlikheid van die vind van die globale optimum te verhoog.

'n Deeglike literatuurstudie word aangebied ten einde die konsepte in hul wyer konteks te plaas, om verwante of soortgelyke werk voor te lê en die nodige agtergrondkennis aan die leser te verskaf.

Die redenasie agter die konsepte wat ontwerp is en die implementering daarvan word uitgelê. Vier konsepte met verskeie variasies is ontwerp en geëvalueer deur toetse op 'n groot aantal toetsprobleme. Die konsepte word beskou as byvoegings tot 'n basislyn DSO-algoritme, en die impak wat hul toevoeging het, is relatief daaraan geëvalueer. Die konsepte is geassesseer terwyl die klassifiseerder se voorspellings nageboots word, wat dus 100% akkurate OVK-modelle verteenwoordig, en dan met die volledige klassifiseerder aangeheg.

Oor die algemeen het verskeie van die konsepte beduidende afnames getoon in die aantal iterasies wat benodig word op baie van die toetsprobleme. Daar was ook duidelike verbeteringe op sommige van die gelykheidsbeperkte probleme. Sommige probleme of uitdagings wat ervaar was word verduidelik, en voorstelle word gemaak vir die verbetering van enige toekomstige implementasies.

Die mees algemeen belowende konsep se algoritme skuif deeltjies waarvoor die posisiereël se nuwe posisie 'n skuif van die toelaatbare gebied tot die ontoelaatbare gebied sou behels, terug na die benaderde punt waar die partikel die grens sou moet oorsteek.

Die beoogde toepassing is tot optimeringsprobleme waar die evaluering van die doelfunksie en beperkings die berekeningstydperk aansienlik oorheers, soos in groter ingenieursontwerpsprobleme met simulasies. Vir sulke probleme sal die berekeningsbokoste wat deur die bou en outomatiese verstelling van die OVK-modelle bygevoeg word, meeswaarskynlik weglaatbaar beskou kan word relatief tot afname in berekeningskoste verkry uit die vermindering in die aantal funksie evaluasies wat benodig word.

Acknowledgements

I would like to express my sincere gratitude to:

- Prof. G. Venter for initiating this project and his continuous support,
- SANSA (South African National Space Agency) for their financial assistance in 2017 (opinions expressed and conclusions herein, are those of the author and should not be attributed to SANSA),
- the MOD (Materials, Optimisation and Design) research group for their feedback,
- and, finally, my family for their continuous support, in word and deed.

Table of contents

List of figures

List of tables

List of acronyms

CFD Computational fluid dynamics
CV Cross-validation
DoE Design of experiments
EDO Engineering design optimisation
FEA Finite element analysis
MDO Multi-disciplinary design optimisation
PSO Particle swarm optimisation
RBF Radial basis function
SQP Sequential quadratic programming
SVC Support vector classification
SVM Support vector machine
SVR Support vector regression

Chapter 1

Introduction

1.1 Background and motivation

Optimisation involves trying to find the best version of something, the optimum, with regards to some property or properties (objective function) and satisfying some constraints. One can go about finding the optimum by simply trying out all the alternative versions. However, that is not efficient, especially when there are many options to consider. Therefore, it is desirable to either quickly reduce the number of versions to consider or settle for quickly finding an option that is at least an improvement. This is what systematic approaches have to offer.

Systematic approaches can be expert-based, where improvements are made based on knowledge about and experience with the problem or similar problems. Alternatively, it can be algorithm-based, where the problem is cast into standard optimisation problem forms and one/more of many possible 'recipes' or algorithms is/are applied to find the optimum. This is known as numerical optimisation.

Numerical optimisation has become more accessible over the past few decades as computers have become ubiquitous. There are many numerical optimisation methods and method types. Each has its own strengths and weaknesses, numerous different algorithm versions, and specific problem types for which it is intended or suited. Most traditionally arise from mathematics and are very rigorous, but others are derived from observing nature.

Particle swarm optimisation (PSO) is one such nature-inspired, global optimisation method that was introduced in 1995 by Kennedy and Eberhart. The idea is intuitive, fairly simple to implement, makes little to no assumptions about the specific problem, and can be adapted for application to a wide variety of problem types. It is especially useful when other - more specialised and efficient - optimisation methods do not work.

Constrained optimisation problems require finding the optimum while satisfying one or more inequality and/or equality constraints. Unsatisfied con-

straints basically say that a solution is impractical or undesirable for some reason (not feasible), despite whatever its objective function value.

PSO is fundamentally an unconstrained optimisation method and in order to apply it to constrained optimisation problems, it needs to be altered or added to. There have been many such alterations that are used to varying degrees of success and with increasing popularity. However, as is the case with similar methods (e.g. Genetic Algorithms), PSO still struggles with finding optima located on the constraint boundary (edge of the feasible space) or that are equality constrained. This is especially of concern to optimisation problems in engineering, where the solution typically does lie on the constraint boundary.

1.2 Aim

The aim of this research was to find a promising new way to improve particle swarm optimisation's ability to solve non-trivial nonlinearly constrained optimisation problems. The specific interest was in improving its ability to find an optimum located on the boundary of the feasible region and/or its ability to solve equality constrained problems.

The specific idea selected for exploration was to create models of the boundary of the feasible region by using support vector machines, a machine learning technique, for classification and the data the swarm already gathers over the course of a run. Then the algorithm would use that information to influence the swarm's search in a way that encourages the particles' exploration of the boundary, thus increasing the probability of locating the global optimum.

1.3 Research objectives

The formal research objectives were selected as below.

1. Develop a constrained PSO algorithm that improves on PSO's ability to solve certain constrained optimisation problems, especially equality constrained problems and problems with a constrained optimum located on the edge of its feasible region.

2. Investigate specifically the idea of adding SVM modelling of the boundary of the feasible region to augment PSO.

3. Test the algorithm thoroughly on a diverse set of test problems compiled from literature.

1.4 Overview of thesis's content

Chapters 2-4 contain the literature review. Chapter 2 gives an overview of optimisation and specifically looks at how PSO fits into the field. Chapter 3 deals with the details of PSO, common constraint handling methods, and work related or similar to what is proposed here. Chapter 4 introduces the basics of machine learning, looks at Support Vector Machines and details of their use.

Chapter 5 sets out how the problem was analysed, the basic concept generation strategy that was selected and the subsequent concepts.

Chapter 6 looks at the development plan that was followed and the details of the implementation of that plan. It discusses the how and why of the testing of the implementation.

Chapter 7 attempts to convey and discuss the most important findings from testing in a concise manner.

In chapter 8 a conclusion is reached, the work summarised and recommendations are made for future research.

Chapter 2

Overview of optimisation

Optimisation is broad, multi-faceted and applicable to many problems in many fields. This section seeks to give a brief overview of the different optimisation problems, the various numerical optimisation methods, and optimisation as applied to engineering design. The focus is on how PSO fits into this broader context and defining the specific constrained optimisation problem which this work is concerned with solving.

2.1 Optimisation problems

When one is faced with an optimisation problem and elects to use numerical optimisation, one must first classify what type of problem it is and then cast it into a general problem format suitable for that class.

2.1.1 Problem classifications

There are many different types of optimisation problems. Each presents their own challenges and/or goals and require different strategies for solving them. Methods differ based on the problems they are designed to solve.

The classification list in table 2.1 was compiled by drawing primarily from Roy, Hinduja and Teti (2008, Table 1) and Engelbrecht (2007, Chapter 2). A problem belongs to many of these classes at the same time. The classes are based on factors such as the number and types of design variables, the type of constraints, the number and nature of the objective function(s), the degree of uncertainty in the environment of the problem and the specific field or multiple fields to which the problem belongs.

2.1.2 General constrained optimisation problem

This subsection sets out the general problem form for constrained optimisation problems that are continuous and have a single-objective function.

Table 2.1: Types of optimisation problems

Classification of types of optimisation problems		
Design variables		
Number of variables	*Single-dimensional / univariate* *Multi-dimensional / multivariate*	
Types of variables	*Continuous problem (continuous-valued real numbers)* *Integer or discrete problem (only whole/natural numbers))* *Mixed-integer problem (both continuous and integer variables)* *Combinatorial problems (solutions from finite set of solutions)*	
Interdependency (separability of $f(\boldsymbol{x})$)	*Independent-variable* *Dependent-variable*	
Constraints		
Side-constrained / box-constrained (variables have fixed ranges)		
Unconstrained		
Constraint-satisfaction or Constrained	Type of constraints	*Inequality* *Equality*
	Relation to variables	*Linear* *Non-linear*
	Separability	*Separable* *Inseparable*
Objective function		
Number of objective functions	*Single-objective* *Multi-objective / Multi-criterion*	
Nature of objective functions	How it is evaluated	*Quantitative* *Qualitative* *Hybrid*
	Evaluation cost	*In-expensive* *Computationally expensive*
	Optimum	*Uni-modal* *Multimodal* *Deceptive (has false optima)*
	Relation to variables	*Linear* *Quadratic* *Non-linear*
	Continuity	*Continuous* *Discontinuous* *Not-defined outside the feasible space*
Separability of objective functions	*Separable* *Inseparable*	
Time-dependence	*Static* *Dynamic*	
Noise	*Random* *Periodic*	
Uncertain environment		
Without uncertainty	*Reliability-based*	*Robust* *Uncertain*
Specific field related or multi-disciplinary		

Say there is a design that can be modelled as being defined by n number of variables and one wishes to find the values for these variables that give the best possible value for some property, which can be considered as a function of these variables. Then the general, non-linear, constrained optimisation problem to be solved can be formally stated in the following way.

Find:

$$\boldsymbol{x} \in S$$

that minimises/maximises:

$$f(\boldsymbol{x})$$

such that:

$$\boldsymbol{g}(\boldsymbol{x}) \leq 0$$

$$\boldsymbol{h}(\boldsymbol{x}) = 0$$

$$\boldsymbol{x}_l \leq \boldsymbol{x} \leq \boldsymbol{x}_u$$

Thus, with the search space, S, defined as all possible values of the vector of n design variables (design vector), $\boldsymbol{x} = (x_1, x_2, ..., x_n)^T$, below an upper limit, $\boldsymbol{x}_u$, and above a lower limit, $\boldsymbol{x}_l$, find the desired extremum, $\boldsymbol{x}^*$ (best possible solution), of the objective function, $f(\boldsymbol{x})$ (which defines the property of the design which is of interest).

$\boldsymbol{g}$ is the vector of m inequality constraints and $\boldsymbol{h}$ is the vector of r equality constraints ($r < n$) that define the feasible space F, where $F \subseteq S$, which the solution must be a part of to be acceptable. A problem is unconstrained if there are no inequality or equality constraints. Constraints define physical or functional limits on a design.

The objective function can be determined quantitatively from either an analytic equation, mathematical model, experimental data, numerical simulations or metamodel. The objective function can, although less frequently, also be qualitative, e.g. aesthetics, manufacturability.

$\boldsymbol{x}_l \leq \boldsymbol{x} \leq \boldsymbol{x}_u$ are also referred to as side constraints or box constraints and they either keep the problem confined to a manageable search space or make sure that the variable values remain physically realistic. They are included in both constrained and unconstrained problems.

Inequality constraints shrink the feasible space while equality constraints reduce the dimension of the feasible space by 1 (a slice of S). For example, if you add an equality constraint to a two-dimensional problem the feasible space becomes a one-dimensional line/curve or for a three-dimensional problem it reduces the feasible space from a 3-D volume to a 2-D surface and adding a second equality constraint then reduces it to a line/curve (i.e. the intersection of the two surfaces). When mentioning the 'feasibility boundary' what is referred to is where the constraint is active ($= 0$) in the inequality constraints' case or satisfied in the case of equality constraints ($= 0$), thus separating the

feasible part of the search space from the infeasible. The constraint boundary may be complex and the feasible space may consist of multiple disjoint regions.

The global minimum/minima or maximum/maxima is/are the point(s) in the feasible space that give the most extreme (smallest or largest) value(s) the objective function can take for any $\boldsymbol{x} \in F$. While a local minimum/minima or maximum/maxima is/are the point(s) in the feasible space that give the most extreme value(s) for a specific region/subset of the feasible space.

2.2 Optimisation methods

Optimisation methods can be classified in various ways related to how they were derived, to what properties they possess themselves or to the type of problems they are intended to solve. Sometimes they are called search methods applied to solving optimisation problems, as they search the space of possible designs/solutions in hope of finding the best one that is feasible/acceptable.

Because optimisation deals with the computational solution of problems, issues surrounding computational complexity, where problems are classified based on how hard or whether they are possible to solve, and the no-free lunch theorem, become relevant. Simply put, the no-free lunch theorem implies that no method is going to be equally good at solving all problems or, alternatively, that improvement in the ability to solve a specific problem necessarily means a sacrifice in ability to solve another (Roy *et al.*, 2008). Hence the necessity for such a vast array of optimisation methods. Wolpert and Macready (1997) presents a detailed look at the no-free lunch theorem as it affects optimisation.

One major classification of optimisation methods is into global and local optimisation methods. Venter (2010) provides a good overview from this perspective. Local optimisation methods can only find a local optimum and not a global optimum. They are typically gradient-based (require first and possibly second order derivative information), exact, very efficient, require little to no problem-specific parameter tuning, and can solve problems with large numbers of design variables. However, they are incapable of locating a global optimum (unless started in its local neighbourhood), are difficult to implement due to their complexity and have a problem with non-continuous search spaces (e.g. discrete/integer variables, numerical noise). Global optimisation methods (e.g. PSO) possess global properties that increase the likelihood of finding the global or near global optimum. They are typically non-gradient based (so handle non-continuous problems better), require problem-specific parameter tuning (that may significantly alter the results), are only suited to problems with fewer variables (< 50) and can have a higher computational cost.

An algorithm is a problem-solving procedure containing a finite set of instructions in a specific execution order (Cormen, Leiserson, Rivest and Stein, 2009) and is easily presented as a flow-diagram or pseudocode. The instructions can be implemented/coded in many different ways. A heuristic is a

rule-of-thumb and, in optimisation, it is an algorithm that cannot be mathematically guaranteed to find the optimum. A heuristic is problem-specific. Metaheuristics are problem independent generalisations of these ideas, that are typically stochastic (possessing random elements) and the most popular choice of method for global optimisation. For an overview see Beheshti and Shamsuddin (2013) or Boussaïd, Lepagnot and Siarry (2013).

Based on Sörensen, Sevaux and Glover (2018), there are some subtle variations in what is meant by metaheuristics. It varies by level of abstraction, how far removed it is from any specific problem or problem type, or by where the line gets drawn between different types of metaheuristics. Sörensen differentiates between metaheuristic frameworks, that Sörensen and Glover (2013) defines as being "high-level problem-independent algorithmic framework that provides a set of guideline or strategies to develop heuristic optimisation algorithms", and metaheuristic algorithms, an implementation of a framework or frameworks that is slightly lower level and more concerned with solving specific problem types. Sörensen (2015) argues strongly that many supposedly different metaheuristics are fundamentally the same and hide their lack of novelty behind elaborate metaphors.

Sörensen *et al.* (2018) gives an overview of the history of metaheuristics, from our natural human tendency of forming heuristics to its scientific study, by dividing it into separate periods based on how the view and study of them has shifted.

Figure 2.1 shows a simplified view of metaheuristics with types sorted based on similar origin. PSO is nature-inspired and population-based.

PSO is a case of computational swarm intelligence, derived from the movement of a bird flock or fish school (Kennedy and Eberhart, 1995). Swarm intelligence is where simple individual agents react to their environment and by cooperating via direct or indirect communication end up exhibiting distributive collective problem-solving behaviour (Engelbrecht, 2007). It is a case of natural emergent complexity, where complex patterns arise that cannot be easily predicted from components and which may or may not be useful. When the ideas of swarm intelligence are implemented in computing it is known as computational swarm intelligence. Figure 2.2 shows its relation to other fields. PSO, evolutionary computation and artificial neural networks are also examples of natural computing. De Castro (2007) provides a good overview of it.

2.3 Engineering design optimisation

Broadly speaking, the application of optimisation in engineering can be divided into optimum design and optimum control. This work is concerned with the numerical optimisation of the first. However, optimum control problems can

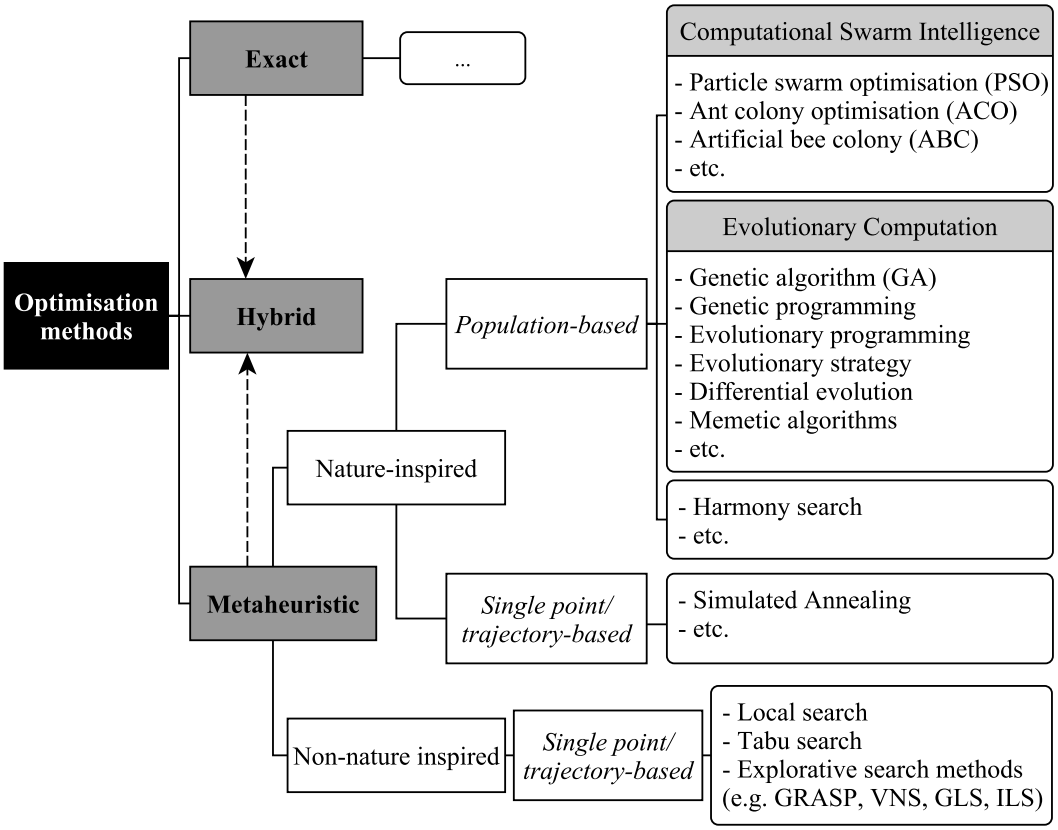

Figure 2.1: Overview of metaheuristics

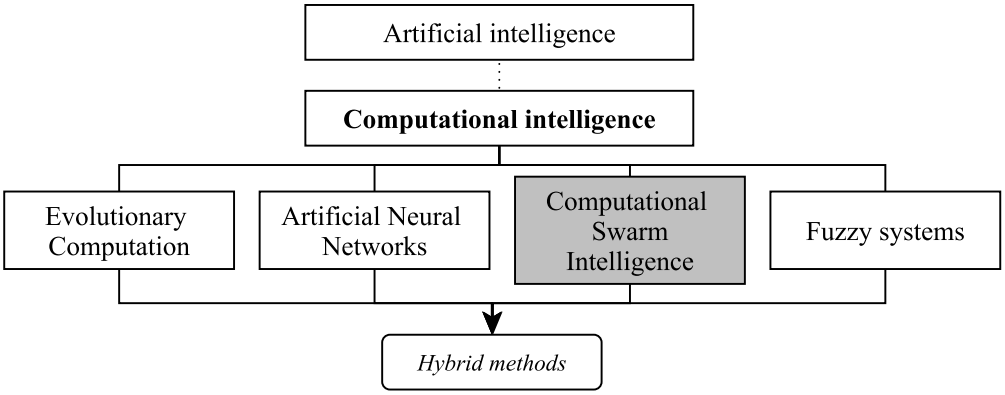

Figure 2.2: Computational intelligence (based on Engelbrecht, 2005, p.5)

typically be transformed into optimum design problems and solved using the same methods.

The conventional engineering design process - a systematic, highly iterative and complex process - has evolved over a long period of time. When intro-

ducing numerical optimisation the process necessarily becomes more formal and rigorous. This can make it more tedious. However, this formalisation of the design problem typically leads to much greater insight into the problem, removes some guess work, facilitates better co-operation when multiple disciplines are involved and ultimately leads to better designs. A good introductory book on the optimum design process is Arora (2004). While Roy *et al.* (2008) provides a good overview of engineering design optimisation.

Engineering design optimisation (EDO) problems have various attributes that distinguish them from other optimisation problems and that may complicate not only casting it into an acceptable problem format, but the selection of an acceptable optimisation method to use and the numerical optimisation process itself. Design evaluation effort and the degrees of freedom of an EDO problem (Roy *et al.*, 2008) are key drivers of decision making on how one handles a specific problem. Furthermore, the reality of how optimisation is applied in industry may differ from in a theoretical or academic environment due to more time, computation resource and software limitations. This should be considered when designing a new method in order to give it the best chance at practical utility.

EDO might be limited to one particular field, e.g. fluid mechanics, and usually involve some degree of computational modelling and simulation, e.g. finite element analysis (FEA), computational fluid dynamics (CFD) or multi-body dynamics. Increasingly, multi-disciplinary design optimisation (MDO) problems are considered and this increases complexity and computational effort.

The objective function and constraints for EDO are typically obtained from simulations that are treated as a black box. Grouping phenomena into a conceptual black box allows for the disguising of complexity, as only its input-output interactions remain of concern (Forsyth and Rada, 1986). It is vital for parallelising computation, which allows for modelling designs and systems in more detail. The point of concern is that this leaves one knowing only single points or 'snapshots' of information scattered throughout the design's search space. This problem is amplified when the evaluation is computationally expensive. A similar problem may exist when dealing with experimental data. This and the fact that real-life problems are often non-differentiable, mean that gradient-based optimisation methods may be unsuitable (Roy *et al.*, 2008).

Metamodels are valuable support tools in engineering design and are especially useful as surrogates for computationally expensive simulations when performing optimisation (Wang and Shan, 2007) or for generalising experimental data. A metamodel is simply a model of a model. Building one involves careful use of an appropriate sampling method for obtaining good data points, choosing a suitable model type and fitting the model to the data by using an appropriate method. However, even a metamodel clearly takes time and effort to construct.

Optimisation techniques may, due to computational time and resource con-

straints, simply be used for design improvements or for constraint satisfaction.

Roy *et al.* (2008) observed that swarm intelligence was increasing in popularity for EDO, and that it could help improve computing efficiency and speed. Tiwari, Hoyos, Hutabarat, Turner, Ince, Gan and Prajapat (2015) did a survey on the use of optimisation in engineering in the United Kingdom. Amongst other things, they found that most commonly the problems that optimisation is applied to are multi-objective, constrained, global and computationally expensive. Metaheuristics, especially variants of evolutionary algorithms and PSO, were found to be popular for solving these.

Chapter 3

Particle swarm optimisation

Particle swarm optimisation has seen much development and been applied in many different fields and to many problem types since its introduction in 1995. Therefore, there are many different ways of presenting it. This chapter attempts to give a broad introduction to the field, focusing on the literature and theory relevant to the implementation (discussed further in chapter 6), how constraints are handled in literature, the problem with optima on the feasibility boundary and the literature on work that is similar or related to what is proposed here.

3.1 Particle swarm optimisation

3.1.1 Background

Particle swarm optimisation, as introduced by Kennedy and Eberhart (1995), applies to the solution of continuous nonlinear problems with an unconstrained search space. They were inspired by the foraging and predator avoidance behaviour exhibited by flocks of birds. Through simple update rules and limited information sharing the particles (points possessing a position, velocity and some limited 'memory') fly through the search space and collaborate to converge on an approximate optimum. They found their method complied with the five basic principles of swarm intelligence. These are the proximity principle, the quality principle, the diverse response principle, the stability principle and the adaptability principle. All these principles deal with how the group of simple individuals react towards and are capable of acting in their environment. The details of the PSO method are discussed in section 3.1.2.

The following general review articles on PSO are a good starting point for those unfamiliar with the field: Parsopoulos and Vrahatis (2002), Hu, Shi and Eberhart (2004), Song and Gu (2004), Poli, Kennedy and Blackwell (2007), Banks, Vincent and Anyakoha (2007), Banks, Vincent and Anyakoha (2008), Poli (2008), Zhang, Wang and Ji (2015), and Bonyadi and Michalewicz (2017).

Also useful are the books by Clerc (2006) and Engelbrecht (2007), and the online bibliography list of the early research till 2004 (Hu, 2006).

Since PSO's introduction, the number of publications per year have kept increasing drastically. Its popularity among researchers is primarily due to its simplicity, from which comes its low barrier to entry and the ease with which it can be combined with other tools and ideas. Also, for example, due to its global properties, lack of use of gradient information, insensitivity to noise and ability to parallelise computation easily. Poli (2008) already wrote of their vast number in his article that looked at publications about the applications of PSO. However, Bonyadi and Michalewicz (2017) found that 75% of the articles on PSO have been published since 2008. Many have merely involved specific applications of PSO and hybridisations of it with other optimisation methods. However, there have been efforts to improve its mathematical theory (Witt, 2011), to analyse the velocity updating rule (Bonyadi, Michalewicz and Li, 2014) and to analyse the swarm's convergence (Schmitt, 2015). All of which is complicated by PSO's stochastic nature (Engelbrecht, 2005). There are some limitations to PSO that have emerged over the years, that Bonyadi and Michalewicz (2017) summarise. These include, for example, problems with regards to local convergence and to stability, and the sensitivity of the search to transformations. However, much of this is of course irrelevant to the average user, who just wants a simple, reliable tool that is at least capable of finding a decent solution.

3.1.2 Basics

To solve a problem using PSO, a swarm of particles (usually 10-50) are randomly initialised within the limits, $\boldsymbol{x}_l \leq \boldsymbol{x} \leq \boldsymbol{x}_u$, of the search space, S, with a particle i receiving a position, $\boldsymbol{x}_i^0$, and velocity, $\boldsymbol{v}_i^0$. The position of each of the particles has its objective function, $f(\boldsymbol{x})$, and any possible constraints, $\boldsymbol{g}(\boldsymbol{x})$ and/or $\boldsymbol{h}(\boldsymbol{x})$, evaluated. This evaluated position is then stored in the particle memory as the best position a particle has seen, $\boldsymbol{p}_i^0$. In the case of PSO with a global neighbourhood, from all of the particle bests the position with the most optimal $f(\boldsymbol{x})$ is selected as $\boldsymbol{p}_g^0$. Then for an iteration k, the particles each have their velocity and then position updated by some simple equations or rules. The new position is subsequently evaluated, compared with the particle best $\boldsymbol{p}_i^{k-1}$ and the best of the two is selected as the new particle best, $\boldsymbol{p}_i^k$. A new best-found solution, $\boldsymbol{p}_g^k$, is then selected from all the particle bests. Compliance with the selected convergence criteria is checked. This may just involve a fixed number of iterations having been executed. If the criteria is not satisfied, then another iteration is performed. By these continual iterations of the particle positions, the swarm eventually converges on the best-found solution. This best-found solution is, hopefully, the global optimum or at least near it.

The velocity update rule for the original PSO from Kennedy and Eberhart (1995) per particle i for each iteration k is as in (3.1), in vectorial form.

$$\boldsymbol{v}_i^k = \underbrace{\boldsymbol{v}_i^{k-1}}_{\text{Previous velocity}} + \underbrace{\frac{c_1 \boldsymbol{R}_{1,i}^k (\boldsymbol{p}_i^{k-1} - \boldsymbol{x}_i^{k-1})}{\Delta t}}_{\text{Particle memory}} + \underbrace{\frac{c_2 \boldsymbol{R}_{2,i}^k (\boldsymbol{p}_g^{k-1} - \boldsymbol{x}_i^{k-1})}{\Delta t}}_{\text{Swarm memory}} \tag{3.1}$$

The position update rule is as in (3.2).

$$\boldsymbol{p}_i^k = \boldsymbol{p}_i^{k-1} + \boldsymbol{v}_i^k \Delta t \tag{3.2}$$

The first term of the velocity update rule is the contribution of the current particle motion, the second term is the influence of the particle's memory and the third is the influence of the swarm's memory or communication. $\boldsymbol{R}_{1,i}^k$ and $\boldsymbol{R}_{2,i}^k$ are diagonal matrices, with different random numbers drawn from an uniform distribution on the diagonal. The factors c_1 and c_2 are the cognitive and social parameters. They dictate the relative influence of either the particle's own memory or the swarm's memory. The time increment, Δt, is assumed to be one and generally omitted from both rules. Differences in the velocity update rule is one of the most common ways in which PSO implementations vary.

The balance between exploration and exploitation or the diversity of the swarm is an important concept for PSO. It is necessary to explore the search space properly (i.e. particle motion is more diverse) in order to increase the chances of finding the global optimum, before converging on a solution or exploiting the knowledge the swarm has gained. The degree to which this happens is influenced by the relative sizes of the three terms. Therefore, changing the sizes of the terms can be viewed as varying the global search capability (exploration) versus the local search capability (exploitation) of the algorithm. Where the previous velocity term encourages more global search and the two memory terms lead to local search around the best-found solutions.

Shi and Eberhart (1998) identified that one could better manage that trade-off, and hence get improved performance of the PSO, by introducing an inertia factor w to the velocity term. It is now the most common version of the PSO algorithm and sometimes is referred to as the standard PSO (Bonyadi and Michalewicz, 2017), as shall be done in this thesis. The velocity update rule for it is as in (3.3).

$$\boldsymbol{v}_i^k = \underbrace{w\boldsymbol{v}_i^{k-1}}_{\text{Inertial velocity}} + \underbrace{c_1 \boldsymbol{R}_{1,i}^k (\boldsymbol{p}_i^{k-1} - \boldsymbol{x}_i^{k-1})}_{\text{Particle memory}} + \underbrace{c_2 \boldsymbol{R}_{2,i}^k (\boldsymbol{p}_g^{k-1} - \boldsymbol{x}_i^{k-1})}_{\text{Swarm memory}} \tag{3.3}$$

Bansal, Singh, Saraswat, Verma, Jadon and Abraham (2011) presents a comparison of the various strategies that have been attempted for determining w. This includes, for example, using either a constant value, a random value, an adapting value, an increasing or decreasing value, or a chaotic value.

The other common version of the PSO algorithm is what some call the linear PSO. Its velocity update rule is as in (3.4).

$$\boldsymbol{v}_i^k = w\boldsymbol{v}_i^{k-1} + c_1 r_{1,i}^k(\boldsymbol{p}_i^{k-1} - \boldsymbol{x}_i^{k-1}) + c_2 r_{2,i}^k(\boldsymbol{p}_g^{k-1} - \boldsymbol{x}_i^{k-1}) \tag{3.4}$$

It is the same as the previous versions except that, as seen by comparing (3.3) and (3.4), for $\boldsymbol{v}_i^k$ instead of multiplying terms two and three with a random diagonal matrix (e.g. $\boldsymbol{R}_{1,i}^k$) one multiplies by a single random value (e.g. $r_{1,i}^k$). Thus, the possible directions that $\boldsymbol{v}_i^k$ can point in is reduced, because the direction of the vector (determined by the subtraction of the vectors between brackets) becomes fixed. This is due to the random value just scaling the magnitude of the term's contribution to $\boldsymbol{v}_i^k$. Which makes the linear PSO easy to visualise with vector addition and thus much easier for beginners to understand. See its visualisation in figure 3.1.

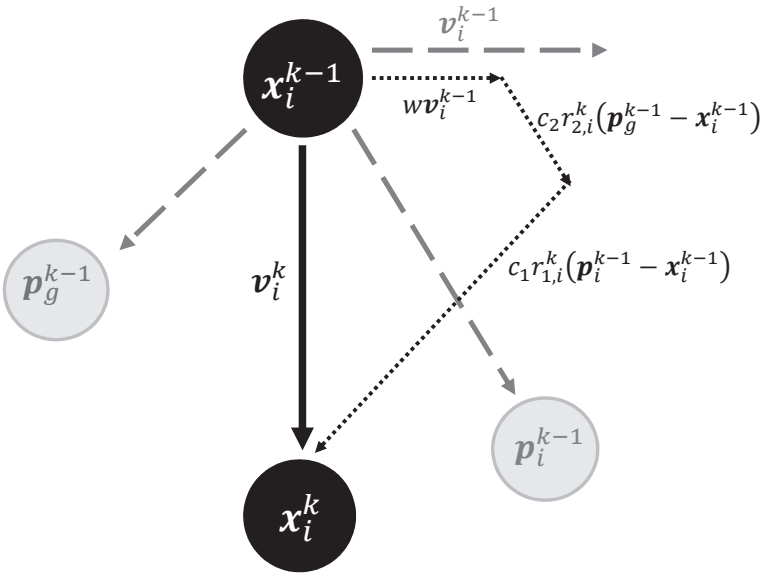

Figure 3.1: Linear PSO velocity update rule visualisation (based on Hassan *et al.*, 2005, fig. 1)

Linear PSO is seen by some as a common error (Clerc, 2006, p. 44) (Bonyadi and Michalewicz, 2017). It is one that is easy to make when you write the equation in vectorial form, due to the poor formatting quality, etc. of the original articles, but especially the fact that the equation is often given per design variable. However, as shown by Wilke, Kok and Groenwold in 2007*a* and 2007*b*, they both have their benefits and limits. The standard PSO possesses much greater diversity, but is not rotation invariant. While the linear PSO is translation, scale and rotation invariant, but may, under specific conditions, reduce a particle's motion to a line search. On a side note, many

sources do not explicitly state that they are using one or the other. Therefore, one should be aware of that when reviewing literature.

A constriction factor K was introduced by Clerc (1999) and improved by Eberhart and Shi (2000) as in (3.5).

$$\boldsymbol{v}_i^k = K[\boldsymbol{v}_i^{k-1} + c_1\boldsymbol{R}_{1,i}^k(\boldsymbol{p}_i^{k-1} - \boldsymbol{x}_i^{k-1}) + c_2\boldsymbol{R}_{2,i}^k(\boldsymbol{p}_g^{k-1} - \boldsymbol{x}_i^{k-1})] \tag{3.5}$$

$$\text{with} \quad K = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \quad \text{and} \quad \phi = c_1 + c_2, \ \phi > 4$$

It is equivalent to the standard PSO in (3.3), but it just produces specific values for w, c_1 and c_2. The canonical values for these are usually $c_1 = 2.8$ and $c_2 = 1.3$ (Carlisle and Dozier, 2001). In general, the values of w, c_1, c_2 and the number of particles can have a clear influence on how well the PSO algorithm solves a specific problem. If possible, some tuning is advised and, depending on the problem type, decent values for these parameters should be obtainable from literature. For further reading see Bonyadi and Michalewicz (2017), which provides a good overview of the literature related to selecting the parameters and understanding the finer details of their influence.

The topology of the swarm dictates how information is shared between the individuals. The other individual particles that a particle shares information with is known as its neighbourhood. This neighbourhood is typically based on the index number, not actually proximity within in the search space, although there have been researchers who have attempted to do this (it is computationally more involved) (Engelbrecht, 2007). The information that is shared between the particles is the best position a particle has found thus far. Then from the neighbourhood is selected the overall best-found solution that is to be used in the social influence term of the particle's velocity update rule.

For a basic PSO a global neighbourhood topology is used. This means the global best solution found is used to influence each particle's motion. It is known as a star social network structure. Each particle is effectively in 'communication' with each of the others and thus each responds to the global best-found solution. Other structures are known as local neighbourhoods. For these the best of the neighbourhood to which the particle belongs, $\boldsymbol{p}_l^{k-1}$, replaces $\boldsymbol{p}_g^{k-1}$ in the velocity update rule. (Engelbrecht, 2007)

PSO as described up until now performs what is known as synchronous updating of the global best, $\boldsymbol{p}_g^{k-1}$. Where the global best is updated only after all the particles have had their positions updated. Alternatively, asynchronous updating involves updating the global best as soon as a new particle has found a position that is better than the current global best. This new solution is then used for the rest of that iteration to update the remaining particles. (Carlisle and Dozier, 2001)

Craziness is a way of adding diversity to the swarm. It is similar to when in nature a bird randomly breaks from the swarm and flies in another direction. This promotes further exploration by re-assigning the velocity vector of

a particle under specific conditions. (Schutte and Groenwold, 2003) (Venter and Sobieszczanski-Sobieski, 2004)

3.2 Constraint handling methods

Metaheuristics share similar methods and challenges with regards to constrained optimisation. PSO's constraint handling is performed and classified similar too evolutionary computation techniques, which is, in turn, generally implemented using methods borrowed from gradient-based optimisation. Eiben and Smith (2007), Kramer (2010), and Mezura-Montes and Coello Coello (2011) provide good overviews of constraint handling from an evolutionary computing perspective. Some general review articles specifically on constraint handling in PSO are Aziz, Mohemmed, Alias and Aziz (2011), Helwig, Branke and Mostaghim (2013) and Jordehi (2015). The challenge for a more serious user is selecting a constraint handling method that seems appropriate for their specific problem.

Some of the earliest work on PSO for nonlinear constrained optimisation was done by Hu and Eberhart (2002). They implemented a strategy of preserving feasibility by allowing particles to explore both the feasible and infeasible regions, but only allowing feasible solutions to become best-found solutions for either the particles or the whole swarm. This worked well on simple test problems.

Penalty methods, which are borrowed directly from gradient-based optimisation, convert constrained optimisation problems into unconstrained ones. They typically do so by adding to the objective function a term where the method is penalised for any constraint violation. They are by far the simplest and most popular constraint handling method and they do generally, maybe with some tuning of the parameter(s), give decent results. Various penalty methods exist. The main ones being static, dynamic, adaptive, annealing and death penalty.

Another method from gradient-based optimisation is the augmented Lagrange multiplier method. Sedlazek and Eberhard (2006) used this method, which is similar to an extended non-stationary penalty function method, with PSO successfully. They found that it had less sensitivity to ill-conditioning (small position change produces large objective function value change) of problems, introduced a small amount of additional computation initially, the multipliers provided a measure of the cost of the constraints, and that their algorithm was a competitive approach for solving small and medium-sized inequality and equality constrained problems in engineering.

Venter and Haftka (2010) converted inequality constrained problems to an unconstrained bi-objective problem for PSO, where the second objective represents the constraint violation. Without the need for parameter tuning, it provided performance comparable to the penalty method.

There are many other interesting methods. There are methods that attempt to repair infeasible solutions. Methods that make clear distinctions between feasible and infeasible points. Multiple swarms, one to account for the objective function and the other for feasibility, have also been used. Various methods to improve PSO's performance on constrained problems by hybridisation with other optimisation methods have been created.

3.3 Optima on the boundary of the feasible space

Schoenauer and Michalewicz (1996) observed that the global optimum is often on the feasibility boundary and that algorithms often fail as they are unable to precisely search the boundary. Thus, the global optimum has one or more constraints active, especially in real-world problems where there are resource limitations, and many algorithms are not truly capable of finding it. Schoenauer and Michalewicz thus believed that it would be beneficial to restrict the search to the boundary. However, their work had simple, known analytically defined constraints. For problem's that rely on engineering simulations, the information about the boundary's location can be harder to obtain. Thus the problem with searching the boundary, is the lack of knowledge about the boundary. Looking at nonlinear programming problems, Michalewicz and Schoenauer (1996) found that the more constraints that are active at the optimum, the more likely algorithms that do search near the boundary are of finding it. Also, searching the boundary is of specific benefit to equality constraints, which are effectively just a boundary.

Based on the research of Venter and Haftka (2010), PSO and other metaheuristics that are inherently unconstrained have a problem with handling equality constraints and optimums located on the boundary of the feasible region. Part of the problem when using a global optimiser is that the feasibility boundary represents such a small subset of the complete search space. Therefore, the likelihood of these stochastic methods encountering it during one search may be low. Thus, the need to interfere in some fashion to ensure that they do.

Handoko, Keong and Soon (2008) remarked that it is quite intuitive to search the constraint boundary for global optima. Lim, Ong, Setiawan and Idris (2010) and Liu, Li, Zhu and Chen (2018) also expressed the belief that knowledge of the boundary of the feasible region would assist in directing the search to these more promising regions of the search space. Bonyadi and Michalewicz (2015) proposed a method for reducing a problem feasible space to just its boundary, so as to encourage searching it. All these proposed methods are discussed in the next section.

Further complications arise when the feasible space is disjoint, meaning

split into two or more regions. Thus, the feasibility boundary is, in fact, multiple boundaries. Therefore, for instance, a too strict confinement of the search to whichever boundary is found first may not allow all the other regions to be discovered. While and Hingston (2013) discuss how infeasible solutions can be useful in increasing the probability of locating the global optimum, especially when the optimum is on the boundary or the feasible space is disjoint. However, they warn of the risk of wasting function evaluations and trapping the search in local optima away from the feasible space. Bonyadi and Michalewicz (2015) proposed a PSO version borrowing ideas from niching that uses multiple sub-swarms to locate these regions successfully. The strength of these sub-swarms lay in their lack of influence on each other. Sun and Jin (2015) also found improvements at finding disjoint regions when using a master-slave sub-swarm PSO.

The general PSO review article, Bonyadi and Michalewicz (2017), found that there had, at that point, not been many attempts to modify PSO to search the boundary. Thus, one can locate the feasible regions with the right PSO version and the problem that still remains is searching their boundaries properly.

3.4 Related and similar work

Particle swarm optimisation has been used in conjunction with machine learning since it was first introduced by Kennedy and Eberhart (1995). They applied it to the training of a neural network in order to find appropriate weights. Paquet and Engelbrecht (2003) used PSO to train support vector machines by solving the required constrained quadratic programming problem. PSO is well-established as a way to train machine learning models and research on using it to help SVM (PSO-SVM) is still being published, e.g. Demidova, Nikulchev and Sokolova (2016). More recently, along with other methods for constructing a metamodel, it is becoming more common to use these methods to assist the optimisation process itself.

Sun, Jin, Cheng, Ding and Zeng (2017) provides a good overview of the use of surrogates or metamodels to assist swarm optimisation and evolutionary computation with the solution of problems that have computationally expensive function evaluations, say time-consuming computer simulations. They found that the most common methods that are used to create surrogates are response surface methodology , support vector machines (regression), artificial neural networks, radial basis function networks, and Kriging. These are apparently typically used to create a global model of the objective function (and possibly constraints) for the whole search space before optimisation and then the optimisation algorithm uses this surrogate for the function evaluations. In higher dimensional spaces or spaces that are more complex to model, local surrogate models have been used to create approximations of sub-regions of

the search space, as needed. The benefit being that local models are more likely to produce an accurate approximation of the search space, for that region. However, global surrogates may serve to filter off noise and local optima, thereby accelerating optimisation (Wise, 2008). Thus, Sun *et al.* (2017) also report that global and local surrogate models have been used in ensemble to take advantage of these properties. Often with either hybrid methods where a combination of a global optimiser and local optimiser are used or, in the case of PSO, the global model is used first and then as the swarm converges it switches to using a local model.

Sun and Jin (2015) used this ensemble method with success for PSO, especially on multi-modal problems. Sun *et al.* (2017) proposed a method for using surrogate modelling with more high-dimensional problems (50-100 variables) for PSO. Yang, Qiu, Gao, Cai, Jiang and Chen (2018) also used the ensemble approach (RBF and Kriging) to assist PSO for expensive black-box problems and found it promising for practical engineering problems.

Regis (2018) found that most literature on surrogate-assisted PSO are only on box constrained problems, thus only the objective function's evaluations are expensive and approximated by surrogate and not the inequality or equality constraints. However, this has been done for other optimisation methods such as evolutionary computation (e.g. Wang, Yin, Yang and Sun, 2018). Regis then proposed a promising new RBF-surrogate-assisted PSO for computationally expensive constrained black-box optimisation that approximates both the objective function and the constraints for function evaluations during optimisation. This built on the work by Regis (2014) for unconstrained problems which built RBF surrogates of the objective function, whose prediction's enabled testing multiple new possible positions for each particle and selection of the predicted best one as the particle's position to be evaluated for that iteration.

Haftka, Villanueva and Chaudhuri (2016) looked at surrogate-assisted global optimisation from the perspective of parallelising its computation and balancing these global optimisation methods' need for exploration and exploitation in how the modelling is incorporated. They found that the relatively new field of surrogate-assisted global optimization is becoming more popular. Also, they saw that the overhead cost of model training, tuning and prediction, which may become considerable, has not really been considered in literature.

While conducting the research for this thesis, a handful of researchers' work was found that have utilised SVM or other modelling techniques in some way to approximate the boundary of a problem during optimisation for purposes other than just use as a surrogate during function evaluations. However, all of their work employed a different approach for using these methods than what is proposed in this work.

The use of classification models to assist memetic algorithms have been explored in a few articles. A memetic algorithm combines a global search from evolutionary computation, with points that are then selected by some measure

for refinement through some form of local search, see Neri and Cotta (2012) for a review. The researchers had a similar idea to what helped initiate this research - that, as the constraints divide the search space into a feasible space and infeasible space and thus produces points that belong to two clear classes, the use of a binary classification method makes a lot of sense.

Handoko *et al.* (2008) appears to be the first instance of using classification models, rather than regression models, for assisting optimisation. They also used support vector machines for performing the classification in their memetic algorithm, but only the simple linear version. Each particle in the population of the genetic algorithm based local search had its local neighbourhood evaluated on a distance basis and if it was a mixed neighbourhood, thus had both feasible and infeasible points, the feasibility boundary was approximated by the SVM. Any point of that neighbourhood that lay on the margin-of-separation between the two classes, and thus is closest to the boundary, was then selected as a starting point for the sequential quadratic programming based local search. They argued, as is done for this work in section 5.1, that the reduction in the number of function evaluations would mitigate the computational overhead added by building the SVM models. Their work only applies to smaller inequality constrained problems that have analytical gradients.

Handoko, Keong and Soon (2009) extended the ideas from the previous publication to problems with only one equality constraint. They remarked on how the problem with equality constraints is that they reduce the size of the feasible region considerably, see section 2.1.2. Their work used the SVM's raw decision function value to select individuals that are near to the feasibility boundary for further local search. They found large reductions in the number of function evaluations required for their simple test problems.

Handoko, Keong, Soon and Chan (2011) present a follow-up, still with only one equality constraint, but with an algorithm that uses the decision function values in a slightly more complex manner. They concluded that the computational overhead that SVM modelling introduces may be inappropriate for the purposes of the small computationally inexpensive problems that they used and suggested the use of less expensive classifiers.

For their memetic algorithm Lim *et al.* (2010) used the classifier's misclassification of training points as a sign that this region of the space was not well known and was thus worth exploring for a possible optimum. Therefore, the misclassified points were selected as the starting points for further local search, in order to improve the knowledge of that region of the boundary and hopefully find a better solution. Their method proved effective for solving their general complex real-world design problem and benchmark problems. They initialised the algorithm using a Design of Experiments technique (section 4.3). Despite using the DoE techniques for initialisation, they saw, as this research also found, that there still may not be at least one point from both the feasible and infeasible regions, especially if the feasible region is a small percentage of S. Not having points from both classes is clearly problematic when attempting to

train a binary classification model. Therefore, they used their artificial neural network based classifier when available, but reverted to the standard practice of using randomly selected points for further local search as needed.

Basudhar, Dribusch, Lacaze and Missoum (2012) presented constrained efficient global optimization (EGO) using probabilistic support vector machines for a classification model of the boundary and Kriging for the objective function. Some of the benefits of using SVM included it being able to use a single model to define all the constraints that form the boundary (similar to what is done later in this work) and that the SVM classifier could handle dependent, discontinuous and binary constraints.

Gao, Sun, Zeng and Xue (2015) presents the only case that was found of using an SVM classifier to assist PSO. They used it to approximate the boundary for computationally expensive problems, as is proposed here. However, they merely used it as a way to test possible new particle positions for constraint violation before evaluation, in order to avoid evaluating infeasible solutions and, so doing, avoid the additional computational cost.

Bonyadi and Michalewicz (2015) proposed a number of functions to reduce the feasible space of a constrained problem to just the region along the boundary. They defined the boundary as points that are feasible with at least one constraint active. They used PSO to test these, finding that the difficulty lay in first finding the boundary, especially if the selected width of the boundary region was very narrow. They proposed using their ideas in some type of adaptive method.

Liu *et al.* (2018), similar to this research, thought that most of the constraint handling for PSO up until now has ignored the fact that the global optimum is typically located on the feasibility boundary, especially for engineering optimisation problems. Similarly, they had the goal of enhancing the search of the boundary. They expressed a similar opinion to what is discussed in chapter 5.1, in that a key issue with having the PSO explore the boundary, is having the algorithm find it. They performed this search by splitting the optimisation into two branches - a penalty function constrained standard PSO performs a global search and a Subset Constrained Boundary Narrower (SCBN) function (from Bonyadi and Michalewicz, 2015) is used to determine the particle closest to the boundary which is followed by a sequential quadratic programming local search from that point in hopes of finding the global optimum or a better best-found solution. This, of course, influences the PSO's movement, directing it more towards exploring the boundary. This work is conceptually similar the work discussed above for Handoko *et al.* (2008) concerning memetic algorithms. However, the global search is different, it is applicable to a larger group of problems and the method for determining whether a point is close to the boundary, and thus selected as a starting point for the SQP local search, is different. Liu *et al.* found that, on both their analytical test problems and engineering problems, the method increased the chances of finding the global optimum.

Chapter 4

Machine learning

This final chapter of the literature review, provides an introductory overview of machine learning techniques, looks at support vector machines as needed for understanding the rest of the thesis, and discusses how training, tuning and accuracy assessment is performed for machine learning models.

4.1 Overview of machine learning

Machine learning techniques are automated methods for performing data analysis (Murphy, 2012). By detecting patterns in or fitting patterns to the known data, they are used to predict future data under uncertainty. The ability to generalise to provide at least a reasonable guess for any possible input, provided the model is trained well, is an important feature of machine learning. The focus is shifted from finding the correct function from input to output, to mimicking the outputs with minimal error. (Marsland, 2009) (Cristianini and Shawe-Taylor, 2000)

It originated from the quest for artificial intelligence, but machine learning developed into a separate field focused on solving more practical problems, with closer ties to fields like statistics, probability, optimisation and metamodelling.

Machine learning algorithms are classified according to how they learn from the supplied training data. The main classifications are supervised learning, unsupervised learning and reinforcement learning. (Murphy, 2012)

During supervised learning example inputs with the expected output are provided to the algorithm and it must infer the general 'rule' from these so it can give the correct output for any possible input. This mapping from input to output is known as the target function and estimating it is the solution of the learning problem (Cristianini and Shawe-Taylor, 2000). Supervised learning is the most commonly used type of learning (Marsland, 2009).

For unsupervised learning, no correct outputs are given. Rather, the algorithm seeks to group inputs together based on perceived similarities. This is called clustering. In reinforcement learning the algorithm is only informed

when it is in error, not what the correct answer is. It must explore options until it succeeds in finding the correct answer.

Classification and regression are the two abilities provided by supervised learning. Often the same algorithms can be modified to perform one or the other. Classification is the ability to deduce to which pre-specified group or class something belongs, or whether or not it belongs to a group. The classic example is deciding whether an e-mail is spam or not. Regression, as with regular functions, is the ability to extrapolate or interpolate from known data to give the expected value of a new point. Thus classification deals with discrete-valued predictions, while regression is concerned with continuous-valued predictions.

4.2 Support vector classification

4.2.1 Overview

Support Vector Machines (SVM) is a popular supervised machine learning technique that can conduct classification or regression. Support vector classification (SVC) was the starting point while support vector regression (SVR) was an extension of SVC to the solution of regression problems (Murphy, 2012).

There are different versions of either SVC or SVR that all have slightly different mathematical formulations and parameters that control how the models fit the data. These include C-SVC, ν-SVC, ϵ-SVR and ν-SVR. SVC can also be adapted to not only work for binary classification, but for one-class and multi-class classification as well. SVC can classify non-linearly separable datapoints, and SVR can approximate nonlinear functions, through the implementation of the kernel trick, which is discussed in section 4.2.3.

4.2.2 Basics

A hyperplane is a sub-space of a n-dimensional space with co-dimension 1 or dimension n-1. The basic idea of binary support vector classification is linearly separating data points, $\boldsymbol{x}_i$, belonging to two clear classes, represented by y_i equal to either $+1$ or -1, by using a hyperplane, $\boldsymbol{w} \cdot \boldsymbol{x}_i + b = 0$. However, there is not an unique solution for this, as many hyperplanes could separate the points. Thus, one needs to obtain the most optimal one.

The most optimal hyperplane is a hyperplane that is placed as far away from the nearest data points on both sides as is possible. It is called a maximal margin hyperplane. Figure 4.1 visualises a hyperplane linearly separating points belonging to two classes in the center of the margin which separates the hyperplane from the two classes. The margin's boundaries are defined by $\boldsymbol{w} \cdot \boldsymbol{x}_i + b = +1$ and $\boldsymbol{w} \cdot \boldsymbol{x}_i + b = -1$. The support vectors are the points on the boundaries of the margin which determine the placement of the hyperplane.

$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$ is called the decision function. The sign of it for a specific point dictates what class the point belongs to.

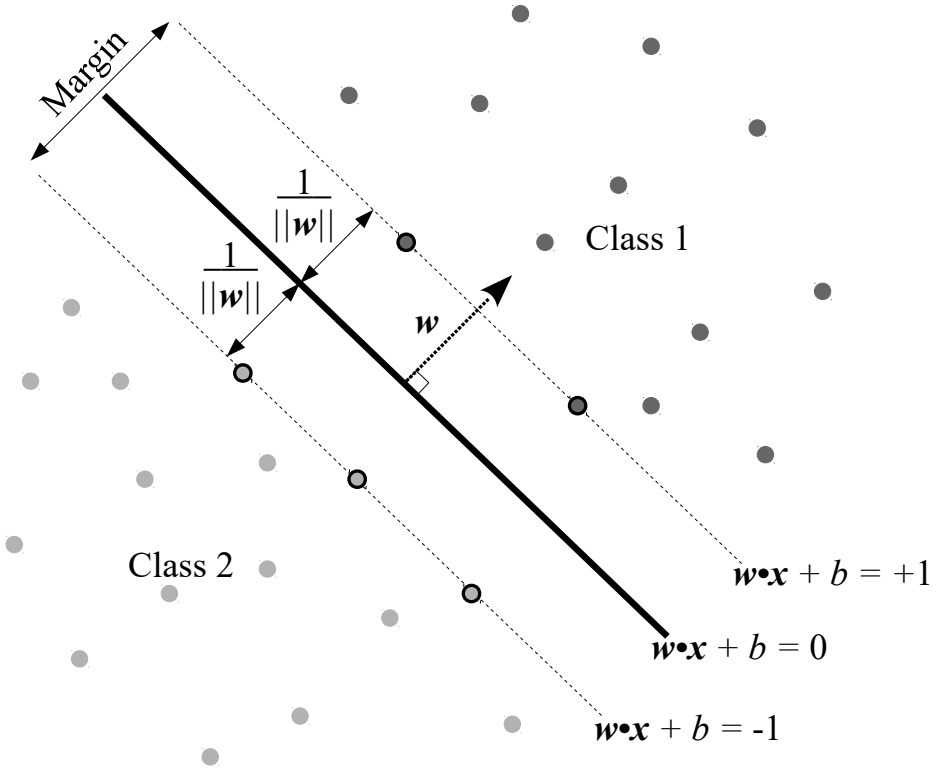

Figure 4.1: SVM model separating two classes of points

The C-SVM is a common soft-margin version of SVM. Soft-margin SVM allows for some points to be misclassified when fitting the hyperplane through the use of slack variables. This helps when the classes are not perfectly linearly separable. Which is opposed to the hard-margin version discussed previously. In order to fit a C-SVM model to training data, the error function given in (4.1) must be minimized subject to the constraints in (4.2). This is typically done through conversion of this primal problem to the dual problem and then using sequential minimal optimization to solve it.

$$\frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{N}\xi_i \tag{4.1}$$

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, \ \ i = 0, 1, ..., N \tag{4.2}$$

N is the number of training data points, $\boldsymbol{x}_i$, with labels for their class as either $y_i = +1$ or $y_i = -1$. ξ_i is a slack value on a data point i, thus how much it may be misclassified in the event that the classes are difficult to separate by the hyperplane. $\boldsymbol{w}$ is the vector of coefficients for the data points and also the

normal vector that controls the slope of the hyperplane. b is a constant which controls the hyperplane's position in the space, that is a distance of $\frac{b}{||\boldsymbol{w}||}$ away from a hyperplane of similar slope through the origin of the space. The user set parameter C controls the influence of misclassification in (4.1). A larger C implies fewer points may be misclassified and results in a smaller maximal margin for the model.

A C-SVM model was fitted in this research using the LIBSVM Python library (Chang and Lin, 2011). Thus the finer details with regards to deriving these equations and how this primal problem is typically converted to the dual problem and implemented, is outside the scope of this work.

4.2.3 Kernels and the high-dimensional feature space

The use of the kernel trick is part of what gives SVM its more general powerful abilities. The addition of a kernel enables SVM to approximate a large variety of classification boundaries where the classes are not linearly separable in the original space. For a good explanation of how the kernel trick was mathematically derived using Reproducing Kernel Hilbert Spaces, see Daume III (2004). Murphy (2012) provides an overview of how it is more generally used in machine learning.

Essentially, a kernel implicitly maps a problem to a higher dimensional vector space called the feature space. Where, in the case of SVC, the classes can in fact be linearly separated, with some error, using a hyperplane. It is an implicit map, as instead of explicitly mapping the data points to the higher dimensional space and then fitting the hyperplane, it only calculates the needed dot product between data points in this new feature space through the use of some function $K(\boldsymbol{x}, \boldsymbol{x'})$. This function is equivalent to the dot product and is also called the similarity between the two points $\boldsymbol{x}$ and $\boldsymbol{x'}$ or, of course, the kernel. The values for the dot product of the support vectors turns out to be all that is necessary to move the calculations for constructing an SVM model's hyperplane to the feature space (when solving the dual problem). The feature space and the explicit map towards it might be highly complex or even unknown. Thus this implicit map reduces the computational expense of mapping the problem to a feature space considerably, while enabling SVM and other machine learning techniques to handle non-linearity in the data. Because of the mapping the kernel introduces, the whole SVM model effectively operates in this higher dimensional space and any values obtained from it exist there as well.

There are many different kernels, the common types used for SVM being the linear kernel, the polynomial kernel, the sigmoid kernel and the RBF kernel. The selected kernel should be appropriate for the data that a model is to be fitted to (section 4.3). The RBF, shown in (4.3), is a good general choice for nonlinear classification. The RBF kernel has only one parameter (γ) that needs to be tuned and fewer numerical difficulties than some of the others but is

unsuitable when there are a high number of classes (Hsu, Chang and Lin, 2008). The RBF kernel effectively implicitly maps the SVM model into an infinite dimensional space. According to Herbrich and Bach (2001) it specifically maps the input data onto the surface of an infinite dimensional hypersphere.

$$K(\boldsymbol{x}, \boldsymbol{x'}) = e^{-\gamma\|\boldsymbol{x}-\boldsymbol{x'}\|^2} \tag{4.3}$$

4.2.4 Inverse mapping from the feature space

Initially, for this project, it was hoped that as calculating the shortest distance between data points and separation boundary is clearly a natural next step from the SVM hyperplane calculations, that it meant that this distance could be obtained and used in some way in the algorithm concepts. Ultimately, the shortest distance from a point to the boundary was used. However, only the distance in the feature space can be calculated easily from the SVM model. Because, as mentioned in the previous section, the use of the kernel trick effectively means that the SVM model's hyperplane is fit to the data in this higher-dimensional space.

It was therefore of interest to this work and subsequently investigated, whether one could invert the map and obtain from the SVM model, in some fashion, the true value for the shortest distance to the boundary in the search space. The map to the feature space, that the kernel performs implicitly, does appear fundamentally to be an isomorphism, based on how it was derived (Daume III, 2004). This usually means that the map possesses an inverse map back to the original space. However, the investigation showed that the mathematics involved is all decidedly non-trivial. It even appears that much is still unknown about aspects of the special spaces that lie behind why kernels work. This is especially true for the more useful kernels.

In the case of the popular RBF kernel, Steinwart, Hush and Scovel (2006) proposed for the first time a method to create an approximation of the explicit map that an RBF kernel implicitly uses. Vempati, Vedaldi, Zisserman and Jawahar (2010) presented a more general finite approximation of the RBF kernel's feature map. Vempati *et al.* state that as the feature map for RBF is infinite dimensional it is not suited for numerical computations, hence the need for approximations.

Approximations such as these for RBF and other kernels typically make use of either the Nystroem method for matrix approximations, the Taylor series expansion or the Fourier transform. They are used with very large datasets. Apparently, the computation cost involved in using kernels does not scale well, because the number of support vectors increases for which the kernel must be evaluated (Fehr, Arreola and Burkhardt, 2008). So one usually reverts to linear SVM when working with large datasets (Hsu, Chang and Lin, 2008). These approximations are thus used to make non-linear learning on large datasets possible.

Therefore, it appears, currently, impossible to obtain, or at least improbable that one can easily obtain in a way that falls within the scope of this research, the inverse map from the feature space. Unless a very simple, and thus not very useful, kernel were to be somehow used for which an inverse map could be found or approximated.

However, the feature space shortest distance was ultimately used as a measure of the distance to the boundary. It is used to facilitate the approximation of the true distance in the search space, as is needed by the concepts presented in chapter 5. More details on how this was done are provided in chapter 6.

4.3 Training, tuning and accuracy assessment

When using machine learning methods, the goal is to use only as complex and computationally intensive a model as is sufficient and no more. Complexity means the ability to represent a wide variety of input-output relationships, which is represented analytically by a model's Vapnik-Chervonenkis dimension. A trained model must represent a good generalisation from the data and not react to small variations such as noise (Murphy, 2012). Therefore, the goal is not to fit all the training data perfectly but to minimize the error in the model's predictions. Underfitted or overfitted models are models that are bad generalisations from the data. The overfitted model has too many degrees of freedom or is too complex, such as using a high-degree polynomial to fit perfectly through all data points, thus producing a zero training error, rather than just fitting a linear line. While an underfitted model has the opposite problem. SVM can, through varying the kernel used and scale parameters, create models of varying complexity. Thus, choosing an appropriate model is of great importance.

Other than the type of model used, a factor that can influence a model's prediction performance is the quality of the data used for training it. That is, whether or not it consists of a large enough set of samples that is representative of the function/phenomena/model that one is attempting to model. Therefore, when one is in a position to choose what data to use, sampling methods borrowed from Design of Experiments are typically used (Wise, 2008). Uniform random sampling can be acceptable, but this does not ensure even distribution throughout the search space unless a large number of samples are taken. Quasi-random methods that result in low discrepancy (so samples are more evenly distributed in S) are preferable, especially when fewer samples are taken. This includes, for example, the Halton sequence (Choset, Lynch, Hutchinson, Kantor, Burgard, Kavraki and Thrun, 2005, p. 220-221), the Hammersley sequence (Choset *et al.*, 2005, p. 222) or Latin Hypercube sampling (Wise, 2008, p. 10).

Models have scale parameters that can be tuned to make the model a better fit for the data. In the case of C-SVM with the RBF-kernel, the two parameters are C for the SVM version and γ for the kernel. Essentially, finding the best

parameters is a simple optimisation problem. It is typically good enough to just use some variation of an exponential grid search (Hsu, Chang and Lin, 2008). A grid with approximately exponentially increasing values is used due to the scale parameters having a multiplicative effect. An initial rough grid search with refinement in a specific area is common, although a simple optimisation method can be used.

Having discussed what makes a well-fitted model and how to go about selecting and constructing a well-fitted model, the question that still remains is - how is this fit assessed practically?

Cross-validation (CV) is generally used to approximate the testing accuracy. This is important as it gives one an approximation of the accuracy (CV testing accuracy) one can expect when using the final model to make predictions (here called prediction accuracy). Having calculated the training accuracy by assessing the number of training points correctly classified by the model, one can assess the fit of the model by comparing the training accuracy with the CV testing accuracy. If the CV testing accuracy is much lower than the (decent) training accuracy, it indicates that the model is overfitted and thus will probably have a low prediction accuracy. In the case of an underfitted model, both the training and CV testing accuracy percentages are similarly low.

There are different types of cross-validation methods. Murphy (2012) presents an overview. The hold-out method, more properly just called a validation method, arbitrarily divides the samples into training samples (say 50-80% of the samples), which are used to train the model, and testing samples (say 20-50% of the samples), which are used to test it. If the error for both groups is similar, then the model is not overfitted. This method is quick and easy, but is prone to sampling bias, due to the arbitrary selection of the two groups.

k-fold cross-validation addresses the sampling bias issue, but with increased computation time. It involves splitting the samples into k number of folds, setting one of those folds aside, training the model on the remaining $k-1$ folds, testing the model on the fold that was set aside, repeating this for each fold and then averaging the results of all the tests to get an expected testing accuracy rate (CV accuracy) for the model. k is typically around 5-10.

The leave-one-out method is simply a k-fold cross-validation where the number of folds equals the number of samples. It is a good way to validate a model but has a very high computational cost.

Chapter 5

Concept generation

There are many different ways in which one could go about accomplishing the aim of this project. Therefore, this chapter contains an overview of how the problem was viewed and analysed, the selected priorities, the general solution strategy that was used and the concepts that were subsequently produced for further implementation and testing.

5.1 Problem definition and decomposition

As stated in section 1.2, the aim was to develop a method for augmenting PSO's ability to solve constrained optimisation problems, specifically with global optima located on the boundary of the feasible regions or that are equality constrained, through the introduction of modelling of the boundary by using the data points already evaluated and support vector classification (SVC). Although other machine learning techniques could also be used for constructing classification models, performing SVC through use of an SVM was selected due to its ability to approximate different types of non-linear functions easily by varying the kernel used (see section 4.2.3). This keeps it as flexible as possible with regards to the problems that can be solved and allows for dealing with the complexity of the boundary when multiple constraints are present.

Section 3.3 sets out the issue of optima on the boundary of the feasible space. The view taken here is that in order to find these global optima on the boundary one would ideally direct the search of an optimisation algorithm towards the boundary and that the main problem with accomplishing this is the lack of knowledge about the location of that boundary. This is what the idea of having a model of the boundary addresses. Provided one can do this with sufficient accuracy, the main challenge shifts to how one uses this new knowledge to affect how the optimisation algorithm explores the search space.

PSO, by how it is initialised and moves, effectively samples data points randomly throughout the search space. The thinking was that, if by extracting more useful information from these already evaluated points, the number

of iterations, and thus function evaluations, needed to find a nearly globally optimal solution could be reduced. Then for problems with objective functions that are computationally expensive, and thus time-consuming to evaluate (e.g. FEA, CFD simulations for engineering design), the overall time needed for optimisation could be reduced. This despite the extra computational overhead that constructing and tuning SVM models add. Alternatively, in the case of constrained problems where no other methods are working satisfactorily, this method could offer a last resort at finding a decent solution and the extra computation time then accepted as the compromise.

It must always be remembered that in the end one does not desire a mere theoretical curiosity, but a tool that will be practically useful. That is ultimately the true measure of success in this case. For it to be practically useful it must not only promise improved results, but it must be simple enough that engineers, scientist, etc. would actually implement and use the idea. After all, some of PSO's appeal lies in the simplicity and intuitiveness of the basic concept and the ease with which it can be implemented. It is this researcher's opinion that many suggested methods in PSO literature are too complex and unsuited to anything but specific niche applications. Thus, apart from the traditional measures of success or improvement when considering optimisation techniques, an idea or concept was dismissed based on whether it added unnecessary complexity.

5.2 Solution strategy

As the goal is to find optima located on the boundary, the swarm must be encouraged to explore the boundary and then allowed to exploit it as per usual, so as not to be left unable to converge on the best-found solution. Therefore, the starting point for concept generation was to assess in which ways a PSO swarm's behaviour can be influenced to this end and what exact information an SVM classification model of the feasibility boundary can provide with which to influence it.

The assessment was that one could potentially influence the swarm's movement in four main ways.

1. Alter the velocity update rule:
    - Add an additional term,
    - Add or subtract some term only sometimes,
    - Multiply the velocity by some factor,
    - Update the particle best,
    - Update the global best,
    - Or occasionally select a particle velocity based on some other rule.

2. Alter the objective function value.
3. Alter the position (e.g. interfere and shift a selected particle's position according to some rule).
4. (Additionally) Use alternative structure to global PSO:
    - Local neighbourhood,
    - Sub-swarm,
    - Leader-follower,
    - Or introduce a local search (similar to memetic algorithms).

The information that is extractable from the models might vary depending on what implementation of SVM one uses. Generally, the information that is obtainable from SVM models was assessed as being:

- the class a position falls in (feasible or not feasible),
- the support vectors (points closest to the hyperplane),
- the support vector's coefficients (their weight or influence),
- the decision function value for a position,
- the distance to the boundary,
    - either in the higher dimensional feature space
    - or in the original search space (by an inverse map back from feature space or through approximation),
- the normal vector to the hyperplane in feature space ($\boldsymbol{w}$),
- the displacement vector (size and direction to the nearest point on the plane),
    - either in the higher dimensional feature space
    - or in the original search space (by an inverse map from feature space or through another calculation).

In order to make these concepts more useful and transferable, thus opening up options for their use, it is better to use information that is also available from other classification techniques, thus class predictions, so that one ultimately obtains more of a black-box type setup. This idea impacted the decisions made with regards to the concepts and the details of how they work to some degree. Part of the original reason for selecting SVM as the classifier was the idea to perhaps use the distance to the boundary in S, which is directly

available from a linear SVM, but a complication is introduced by mapping to the higher dimensional feature space, see section 4.2.4. Thus, one must either use the value in the feature space, which one can do if only some relative measure of distance is needed, or use whatever information the SVM readily provides to approximate the distance in some other fashion. Retaining its full powerful ability to approximate various nonlinear functions, that the kernel enables, is ultimately far more valuable as it allows for the solution of more general constrained problems.

It should be remembered that a model of a boundary is an approximation and not to expect it to be or treat it as an exact representation. Also, the reason PSO and other metaheuristics work is due to their stochastic nature. They basically conceptually lie somewhere between a completely random search and a perfectly deterministic search, which is what enables them to quickly find a near globally optimal solution. The addition of the influence of the SVC model and the concepts should not diminish this by being say too deterministic.

Either a penalty method or the augmented Lagrangian algorithm, which both convert the constrained objective function to an unconstrained one (section 3.2), should ideally still be used. The use of these constraint handling methods is still needed, ideally, as they make infeasible solutions seem worse to the algorithm. The concepts proposed in this work only interfere by directing the swarm's search to the boundary where one believes the feasible global optimum to be, but has no specific mechanism other than an improved objective function value for favouring a feasible solution over an infeasible one if a measure of it is not introduced to the objective function.

Four concepts with some options were finally selected for implementation and testing. They are discussed next along with why certain choices were made, what other variations of them are possible, where they could possibly encounter problems and any similarity some of them may bare conceptually to existing ideas in literature.

5.3 Concept 1

In short, concept 1 shifts particles for which the position rules' new position would entail a move from feasible to infeasible, back to the approximate point they would have to cross the feasibility boundary.

Concept 1 affects the PSO algorithm right after a new position has been calculated for a particle using the position update rule, but before this new position's objective function value is evaluated. Using the SVM model, the proposed new position for the iteration's class is evaluated. If the SVM predicts that the particle would be moving from feasible to feasible, infeasible to feasible or infeasible to infeasible, the new position is accepted, as per usual.

However, if according to the prediction the particle would be moving from feasible to infeasible, the particle is then not moved to the new position, but

rather, the point where the particle would have crossed the boundary is approximated along the vector between the previous position and the proposed new position. This point then replaces the new position that the velocity and position update rules provided. In this way encouraging more exploration of the boundary.

Figure 5.1 attempts to visualise this for a two variable problem. The feasible region is the grey area where all the constraints are negative. As can be seen, the only particle that is affected is the one that would move from the feasible to the infeasible region if the position update rule were obeyed. It is instead shifted back to the approximated boundary. Important to note is that for equality constraints, where is spoken of feasible space, what is referenced is where the equality constraint would be negative, as technically the boundary is its feasible space.

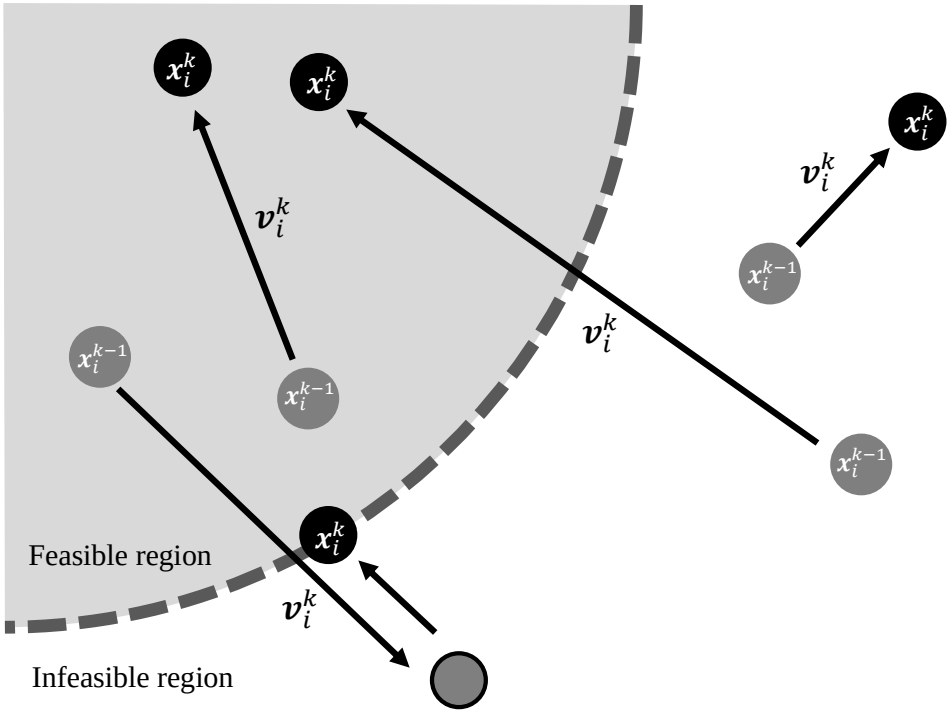

Figure 5.1: Concept 1 visualisation

This boundary crossing point can be approximated using different methods. Five options were considered. Although others are, of course, possible.

1. Linear interpolation using distance to the boundary (C1o1)
2. Golden section search using distance to the boundary (C1o2)
3. Quadratic interpolation using distance to the boundary (C1o3)

4. Bisection search using class predictions (C1o4)

5. Golden section search modified to use class predictions (C1o5)

One possible downside of this concept is that the exploration of the search space might be slightly more limited, as particles are effectively restricted from leaving the feasible region once they have entered it. Or if the feasible space is disjoint, particles might become trapped in one and unable to converge. In this way, it could be seen as somewhat conceptually similar to constraint handling methods that preserve feasibility or repair it. This, of course, assumes a perfectly accurate SVM model, in reality the model is an approximation and may think a particle is remaining in the feasible region, when it has in fact left. So some inaccuracy in the model could actually help reduce the impact on exploration, while also producing new infeasible points to help improve the accuracy of the SVM model.

Another point to consider is that if the SVM incorrectly predicts that the new position will take the particle from the feasible region to the infeasible, and thus the particle is shifted back, the particle will end up further away from the boundary than it could have been.

5.4 Concept 2

Concept 2 attempts to make it more difficult for particles to move further away from the boundary. This is done by reducing the magnitude of their velocity when they are in proximity to the boundary.

The velocity at iteration k from the velocity rule for each particle, $\boldsymbol{v}_i^k$, is reduced by a factor α_i^k, as in (5.1), which is calculated by (5.2).

$$\boldsymbol{v}_i^k = \alpha_i^k \boldsymbol{v}_i^k \text{ with } \alpha_i^k < 1 \tag{5.1}$$

$$\alpha_i^k = 1 - f(d_i^{k-1}) \tag{5.2}$$

$f(d_i^{k-1})$ is calculated by (5.3). d_i^{k-1} is the shortest distance from the particle to the boundary after the previous iteration, $k-1$. While $f(d_i^{k-1})$ is the 'probability' of that distance, as given by the probability density function (PDF) for the (assumed) normal distribution of the particles' shortest distances to the boundary, with its average set to zero. The PDF or bell curve can be seen as centred at the boundary and striving to zero further away. σ^k is the PDF's standard deviation, calculated based on all the particle's shortest distances to the boundary after the previous iteration.

$$f(d_i^{k-1}) = \frac{1}{\sigma^k\sqrt{2\pi}} exp\left(-0.5\left(\frac{d_i^{k-1}}{\sigma^k}\right)^2\right) \tag{5.3}$$

Thus, α_i^k is a function of the distance to the boundary and becomes smaller the closer a particle is to the boundary. In so doing, it shrinks the velocity vector for that iteration and discourages it from moving away from the boundary. Further away from the boundary $f(d_i^k)$ strives to zero, therefore, α_i^k strives to one and $\boldsymbol{v}_i^k$ remains unaffected. As this is a probability distribution, α_i^k could only become zero if all the particles where located at exactly the same point on the boundary, which would mean $\sigma^k = 0$, thus division by zero, resulting in an error. However, this seems unlikely to occur.

The influence that this could exert on the swarm's exploration and convergence is expected to be rather complex and is outside of the scope of this research. As with other heuristics, here it is seen as good enough if it works and provides good solutions.

5.5 Concept 3

Concept 3 introduces sampling centred around the current best-found solution, $\boldsymbol{p}_g^k$. It evaluates which points of those sampled are predicted to be on or near the boundary, to a certain tolerance, and then these points' objective function value and constraints are evaluated. This process is visualised in figure 5.2. If one of the particle's on the boundary is an improvement on the best-found solution, it is updated. Also, the additionally evaluated points are added to the data points for training the SVM. Two different options, C3o1 and C3o2, for conducting the sampling were tested.

C3o1 generates random samples on the surface of an unit sphere and the sphere's radius is increased step by step until at least one of the samples are predicted to be within a certain tolerance of the boundary or a preset maximum number of increases have been reached. Alternatively, C3o2 performs a certain number of random sampling within a n-ball of a fixed radius. Naturally, sampling is only performed if the best-found solution changed during the previous iteration.

In retrospect, both of these options introduce more parameters to tune, which should have been avoided. The maximum radius of C3o1 and the radius of C3o2 could have been automatically set, by using say the distance to the particle that is currently nearest, but not at, $\boldsymbol{p}_g^k$. Another possible variation of this concept is choosing other points, say either the particles' bests or particle's that are within a certain tolerance of the boundary, as centre points for sampling.

This concept uses the SVM model differently than the others. It does not directly alter the position and velocity of the particles, but more indirectly affects it for all particles, via a potential alteration of $\boldsymbol{p}_g^k$ at the start of iteration k, and thus term 3 of the velocity update rule for all the particles.

Conceptually it is not a particularly novel idea, as it is more similar to how surrogates have been used in the past, see section 3.4 where Regis (2014) is

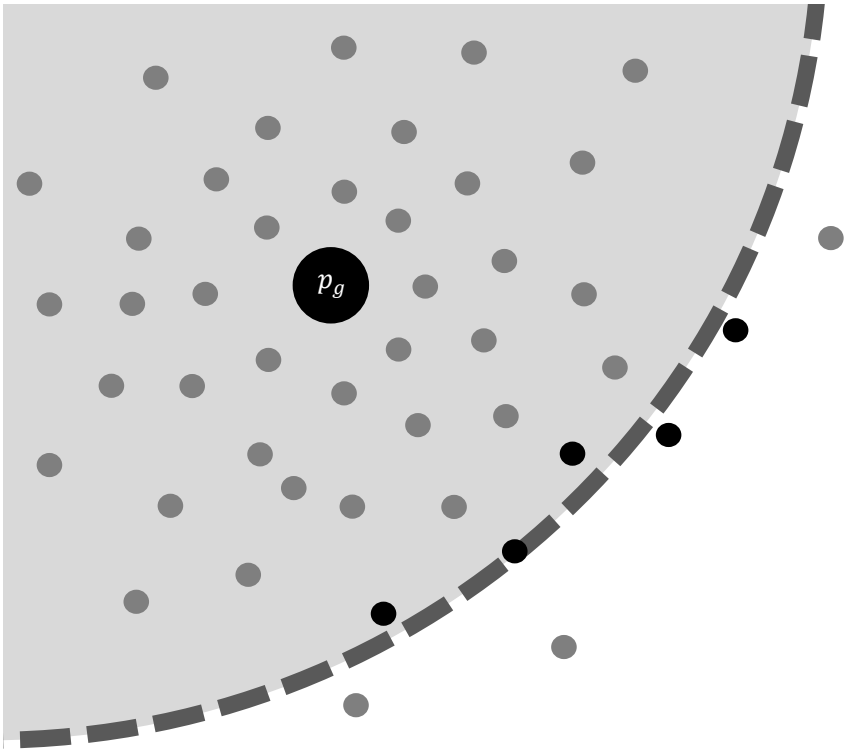

Figure 5.2: Concept 3 visualisation

discussed. However, it is different in that classification models are used instead of regression models. Also, points are selected for proper function evaluation based on whether the model predicts that they are on or near the boundary, rather than whether it is predicted to be an improvement on the current best-found solution, which is slightly similar in idea to Handoko *et al.* (2008) and Liu *et al.* (2018), although without the addition of a local search that just uses an available boundary point merely as a starting point. Admittedly the basic reason for doing that is the suspicion that one of these points very well could be better than the current global best, as the starting point of this argument is that one is looking for an optimum on the boundary. Especially as the measure for 'best' is here the converted to unconstrained via penalty method objective function, so in the case of specifically equality constraints, it quite possibly will be. Also, the swarm is only affected if their function evaluation reveals that one of the points is an improvement on the best-found solution, so all the particles have their motion directly influenced and not just one particle is affected. One could also probably frame this concept as being a case of hybridising PSO with a form of surrogate-assisted local search around the best-found solution.

5.6 Concept 4

In short, concept 4 randomly selects particles that it gives a new velocity vector which is pointed in the direction of the particle's approximate closest point on the feasibility boundary.

Concept 4 is essentially a version of craziness, see section 3.1.2, the new velocity direction a selected particle is given is just not completely random.

Particles are subjected to velocity reassignment with a certain probability, P_r. A selected particle's new velocity is determined by (5.4), which is deliberately kept similar to the format of terms 2 and 3 of the velocity update rules of PSO.

$$\boldsymbol{v}_i^k = c_3 r_{3,i}^k (\boldsymbol{b}_i^k - \boldsymbol{x}_i^{k-1}) \tag{5.4}$$

Where $\boldsymbol{b}_i^k$ is the point on the approximated boundary that is closest to particle i and c_3 is an user set parameter.

Thus, what happens is that the particle is attracted to an approximation of the point that is nearest to it on the boundary. See figure 5.3.

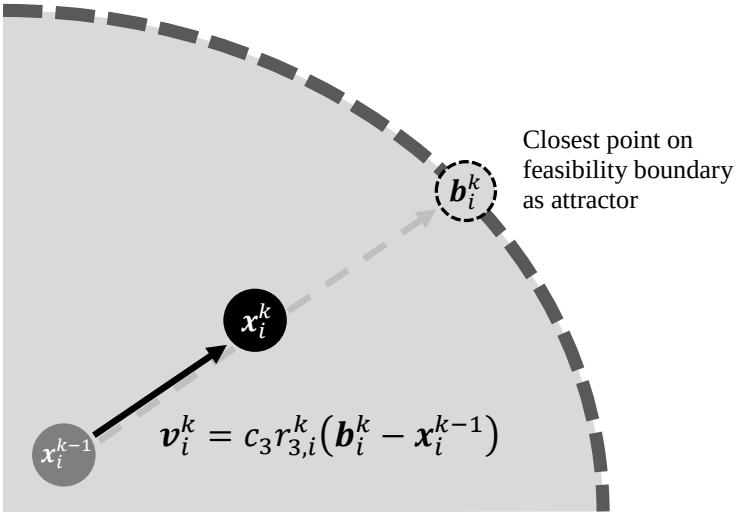

Figure 5.3: Concept 4 visualisation

This concept assumes that the nearest point that is approximately located on the boundary can be calculated. Sections 6.3 and 6.4 presents how this was accomplished for this research. Its calculation is a natural by-product of trying to determine the distance to the boundary for some of the other methods and thus was easy to use here. A variation of this idea could be using any point on the boundary. The original idea involved obtaining one by determining what the particle's closest support vectors are, based on distance, and then performing a line search between them to find an approximate point.

The tunable parameters are c_3 and P_r. In keeping with the other cognitive parameters of the velocity update rule, c_3 adjusts the relative strength of the particle's attraction to, here, the closest boundary point. While P_r regulates how many particles have their velocity reassigned. It should not be too high, else the convergence of the swarm might be negatively impacted if too many particles keep flying off, which is especially of concern if the best-found solution, or maybe even the global optimum, turns out not to be located on the boundary or near it.

A possible variation on this idea could be adding this velocity calculation as an additional term to the update rule. The term could maybe be activated if the particle were to leave the feasible region to draw it back to the boundary. But, apart from the computational cost of calculating multiple boundary points, one would need to check carefully for any possible negative impact on convergence, especially when either the best-found solution or the global optimum is not on the boundary. However, if the boundary point calculation could be performed cheaply enough and the terms influence possibly reduced over time, it could be interesting to attempt.

Chapter 6

Implementation and testing

Having detailed the view of the problem that was taken and the concepts that were selected for testing, this chapter presents how the concepts were implemented for testing, why certain decisions were made and the general testing strategy that was followed.

6.1 Development plan

A step-by-step implementation process was followed in order to break things up into manageable and testable pieces. There were, broadly speaking, three main stages with regards to implementation and testing.

1. **Base PSO:** Implementation of a baseline PSO algorithm and Python program for development and experimentation. To which the concepts were to be eventually added and compared. The base PSO had its performance assessed on various problems types in order to understand its behaviour and have it serve as the baseline to showcase the effect the addition of the concepts have on it. (Section 6.2)

2. **Mimic SVM:** The concepts were added to the existing program, but with a mimicked version of the SVM model. The mimicked model simulated the information to be provided by the SVM, by using the test problems' known constraints. Thus it is in effect an ideal case, where the 'SVM model' is 100% accurate. It also provided initial feedback on the concepts. (Section 6.3)

3. **Full SVM:** The complete implementation with SVM models providing the concepts with the predictions they require. By comparing this to stage 2, the impact of adding the classifier is clear. (Section 6.4)

Each stage of the implementation needed to be tested on test problems. Section 6.4 details the approach that was taken with regards to testing, the

test problems which were selected and some general notes on the difficulty of testing metaheuristics. The results, the comparison of the results of the different stages and the conclusions that can be drawn from that are presented in chapter 7.

All three stages were implemented from scratch in Python2 (Oliphant, 2007) with appropriate libraries, e.g. LIBSVM (Chang and Lin, 2011), Matplotlib (Hunter, 2007) and Numpy (Oliphant, 2006). It was done this way as this gives the greatest insight into what is happening, allows for greater control and provides a learning opportunity. Essentially a development environment was created. A mostly all-in-one program was created which is capable of switching between a wide variety of algorithm versions. Such as different base PSO versions, the concept variations, the mimic SVM and the real SVM implementation's variations. The program allowed for development and expansion as needed, for testing and for outputting a wide-variety of useful data and graphs.

6.2 Base PSO

The baseline PSO algorithm version which was selected to add the concepts to and to compare their performance to, is detailed here. An simple base algorithm with acceptable performance on various continuous single-objective problem types was the goal. A simple base algorithm was chosen to keep it easy to understand, thus more controllable, and to make it more accessible to people working in various specialisations or who want to transfer these ideas to their own (probably more efficient) PSO implementations. The final goal was to be able to inform other researchers that adding the concepts improves this base, so it would be worth attempting to apply it to their own PSO version to see whether it also improves their results.

6.2.1 General overview

The flow diagram in figure 6.1 outlines the algorithm for constrained optimisation problems. In summary, it is a linear PSO with a global neighborhood, asynchronous updating, a constriction factor velocity update rule, with craziness added, and that used the penalty method for constraint-handling. The unconstrained algorithm is mostly the same as the canonical PSO set out by (Carlisle and Dozier, 2001). A fixed number of iterations was used in all testing.

6.2.2 Neighbourhood

This implementation makes use of a global neighbourhood, so that each particle is effectively in 'communication' with each of the others and so responds

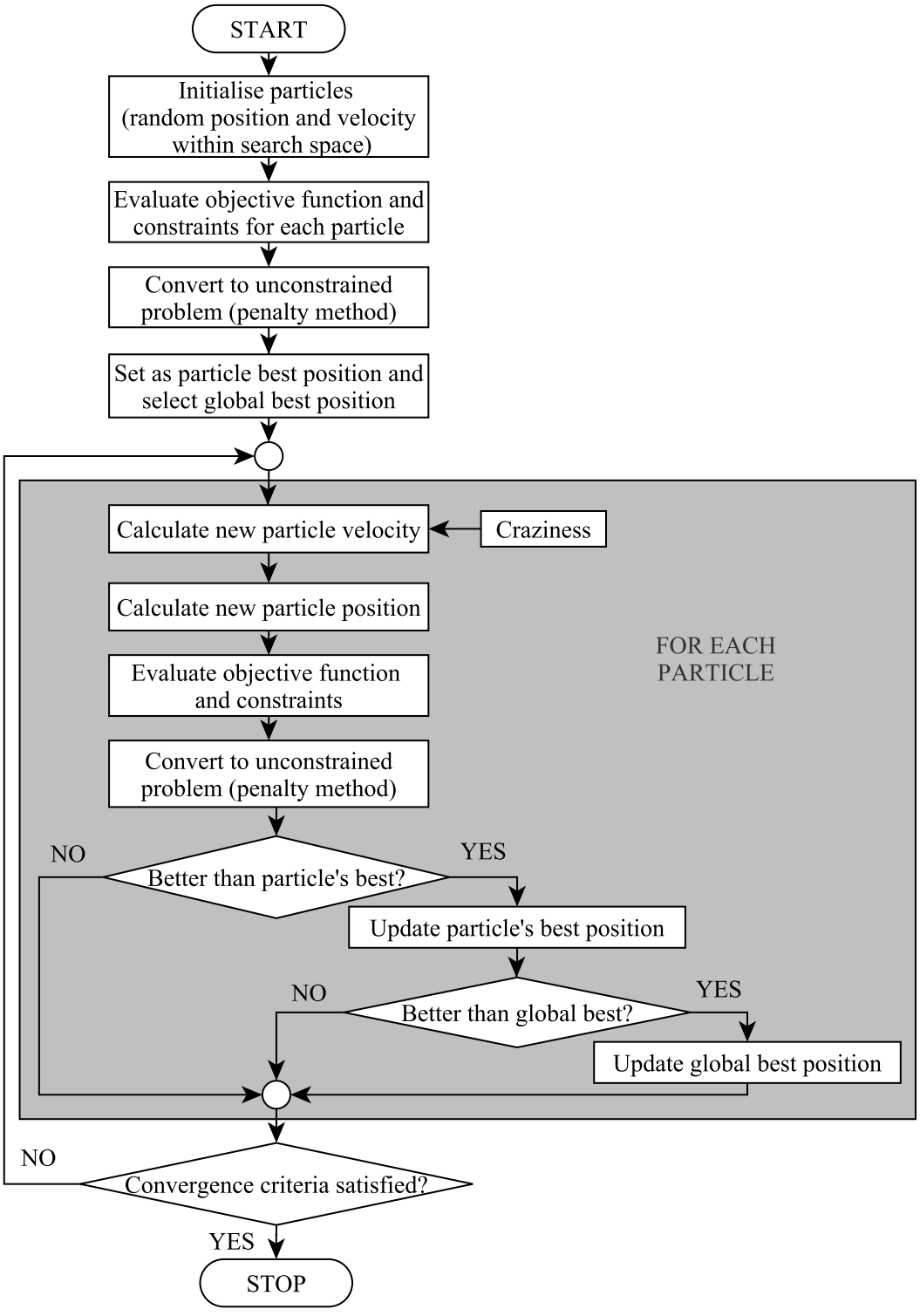

Figure 6.1: Base PSO algorithm flow diagram

to the global best-found solution. The global neighborhood was selected as it is simple, popular and avoids more detail that would need tuning. The global best is here updated asynchronously, thus during one iteration if a particle's new best is an improvement on the current global best, it is updated. In so doing, immediate feedback of the best regions is ensured, not waiting for a new iteration. In general, for the computationally expensive problem types this research is intended for it would be beneficial to rather use synchronous updating, to allow for easier parallelisation of the computations.

6.2.3 Initialisation

The initial starting positions for 30 particles are sampled from a uniform (pseudo) random distribution (Numpy random, seeded with the current time) across S, as defined by the side constraints for each design variable. Each particle is also given an initial velocity, that is sampled from an uniform distribution within the ranges calculated from the limits set by the side constraints of each variable according to (6.1).

$$a = \frac{x_u - x_l}{4} \quad \text{with} \quad v_u = a \quad \text{and} \quad v_l = -a \tag{6.1}$$

6.2.4 Velocity update rule

The velocity update rule for each particle i in the swarm, at iteration k, is a linear PSO with a constriction factor K, see section 3.1.2, with $c_1 = 2.8$ and $c_2 = 1.3$. A linear PSO was selected as it has the advantage that it is simpler to understand its influence on the particles. However, it does decrease the diversity of the swarm. Therefore, some craziness is incorporated, which adds back diversity and so encourages further exploration of the search space by randomly re-assigning a random velocity vector to a particle with here a 2% probability (Schutte and Groenwold, 2003).

6.2.5 Position updates and side constraint handling

The position of each particle i is updated at each iteration k with the standard equation as in (3.2). The side constraints of the search space, here only box-constraints on the variable ranges, are enforced on all position updates, which, for practical problems, maintain the physical realism of the solution. If the value of a design variable exceeds its upper or lower limit the offending variable is altered to the value of its limit (moving the particle back into the search space). The particle velocity is then also set to zero, as the particle's inertia could possibly cause the particle to re-cross the limit in the next iteration (Carlisle and Dozier, 2001). The particle bests and the best-found solution then draw the particles back into S.

6.2.6 Constraint handling

A linear exterior penalty factor, ρ_g (default = 1000), with a step, ρ_{step} (default = 1000), added at the feasible boundary is used for the m inequality constraints. While just a penalty factor, ρ_h (default = 1000), is used for the r equality constraints. No step is used as small violations of an equality constraint is more acceptable. Thus the problem's objective function is converted to an unconstrained one as per (6.2).

$$f^k_{i,unconstrained} = f^k_{i,constrained} + \sum_{j=1}^{m}[\rho_g max(0, g^k_{i,j}) + \rho_{step}] + \sum_{z=1}^{r} \rho_h |h^k_{i,z}| \quad (6.2)$$

The reason for selecting the penalty method is its simplicity and popularity. Note also that particles are not restricted to the feasible region upon initialization or later on. This allows for some further exploration of the search space which is critical in the event that the feasible region is disjoint.

6.3 Mimic SVM version

The second development stage involved adding the concepts to the base algorithm, along with functions to simulate the information they would require from the SVM model.

6.3.1 Simulation of the SVM models' predictions

Simulating the class prediction here was simple. As the test problems have known analytical constraints the constraints were simply evaluated. With the feasible class of $+1$ defined as where all the constraints (both inequality and equality) are negative or equal to zero and the infeasible class of -1 represents everything else.

The prediction of the shortest distance to the boundary for a point is needed by some of the concepts. Determining the distance to a constraint was defined as a simple constrained optimisation problem and then solved using the DOT optimiser (Vanderplaats, 2011). The optimisation problem was to find a point on the constraint that minimized the distance between it and the point of interest (e.g. particle position). DOT was set to use the Modified Method of Feasible Directions (Wise, 2008). The distance to each constraint had to be calculated and the shortest was then selected to be sent to the algorithm. In retrospect, the use of a gradient based optimiser to do this is perhaps not ideal.

For concept 4 the closest boundary point is needed. The same calculations as for the shortest distance was performed, but for this concept, the point on the boundary corresponding to the shortest distance was returned instead.

6.3.2 Concept implementation details

The basic idea of how the concepts alter the base algorithm should be clear from the discussion in the previous chapter. However, some additional flow diagrams of each concept's algorithm, as was implemented for this work, are provided in appendix A. The more important details will be provided here.

For concept 1, the most important detail is the method used for conducting the line search. C1o1 used a standard linear interpolation, C1o2 used a standard Golden Section line search and C1o3 used a standard quadratic interpolation. All utilising whatever measure of the shortest distance to the boundary they were provided with. C1o4 is a standard bisection line search (bracketing method) that uses the sign of the class prediction, where $+1$ is feasible and -1 is infeasible. While C1o5 used a Golden Section line search that was modified so that it also uses the class predictions, similarly to C1o4, as set out in Algorithm 1. A tolerance of 10^{-3} was used for the line searches.

Algorithm 1 Modified Golden Section line search using class predictions

Require: $class(\boldsymbol{x}_i^{k-1}) = 1$ and $class(\boldsymbol{x}_i^k) = -1$
$[a, b] = [0, 1]$ ▷ *a is at $\boldsymbol{x}_i^{k-1}$ and b at $\boldsymbol{x}_i^k$*
$golden = \frac{1+\sqrt{5}}{2}$
$l = b - \frac{b-a}{golden}$
$u = a + \frac{b-a}{golden}$
$eps = abs(b - a)$
while $eps > 10^{-3}$ **do**
    $\boldsymbol{x}_l = \boldsymbol{x}_i^{k-1} + l(\boldsymbol{v}_i^k \Delta t)$
    $\boldsymbol{x}_u = \boldsymbol{x}_i^{k-1} + u(\boldsymbol{v}_i^k \Delta t)$
    Get predictions for $class(\boldsymbol{x}_l)$ and $class(\boldsymbol{x}_u)$
    if $class(\boldsymbol{x}_u) = class(\boldsymbol{x}_i^{k-1})$ **then** ▷ *Thus $l = u = a = 1$*
        $a = u$
    else if $class(\boldsymbol{x}_l) = class(\boldsymbol{x}_i^k)$ **then** ▷ *Thus $l = u = b = -1$*
        $b = l$
    else ▷ *Thus $l = a = 1$ and $u = b = -1$*
        $a = l$
        $b = u$
    $l = b - \frac{b-a}{golden}$
    $u = a + \frac{b-a}{golden}$
    $eps = abs(b - a)$
$\boldsymbol{x}_i^k = \boldsymbol{x}_i^{k-1} + 0.5(\boldsymbol{v}_i^k \Delta t)$

Concept 2 merely has to calculate the factor α_i^k. The standard deviation of the all particles' shortest distances to the boundary after the previous iteration is calculated at the start of the new iteration.

For concept 3 the number of samples to be taken around the global best-found solution was set as four times the number of variables, n, a problem had. A tolerance of 0.1 on the distance was used for deciding whether a point qualified as being on the boundary. For C3o1's sampling on the surface of a sphere, sampling was only allowed twice, first on the unit sphere and then again on a sphere of radius 2. While for C3o2 random sampling was performed inside an n-ball of radius 2.0.

For concept 4 the probability of reassignment, P_r, was set as 10% and $c_3 = 2$ was used. As discussed, these choices of especially the parameters for C3o1, C3o2 and C4 are somewhat arbitrary user-selected parameters and ideally, this should be automatically calculated based on the specific problem or at least some problem specific tuning should be performed.

6.4 Full SVM version

Stage three of the development meant the addition of the support vector machine to the program. Provided here is primarily the final version of the setup, with some discussion of problems encountered and alternatives.

6.4.1 Adding the SVM-classifier

The SVM classifier and its cross-validation were provided by the LIBSVM library (Chang and Lin, 2011). This is the same library that powers the SVM of the Python library that is commonly used for machine learning, Sci-kit Learn. However, using it directly allowed for greater insight into what was happening and for potentially altering it to suit the needs of this work.

A modified version of LIBSVM (Di, 2011) was finally selected for use. It was modified to be capable of calculating distances to the hyperplane. If a linear classification is performed, it produces the real distance from a point to the hyperplane in the search space. However, with the use of a kernel, the distance in the feature space is calculated.

Ultimately this algorithm has the PSO performing the sampling of the search space for the SVM. Thus, concern developed that the uniform random sampling of the PSO's initialisation, especially with regard to even distribution throughout the search space, might not be good enough for this purpose, especially in the early iterations of a run. See section 4.3 for a discussion on how sampling is generally performed for SVM.

Therefore, some initial informal tests were done on constructing a SVM model, comparing the cross-validation accuracy of the model when using the various sampling methods on a handful of test problems. No drastic differences were observed. However, some concern remained and so both the Mimic SVM and Full SVM versions were run not only using the regular initialisation using

uniform random sampling but also using Latin hypercube sampling, performed by the Chaospy library (Feinberg and Langtangen, 2015).

In order to train the SVM, it is necessary to have data with at least one point located in each class. Lim *et al.* (2010) also ran into this issue of ensuring that the classifier had at least one point from both classes, see section 3.4. There exists a one-class SVM, the use of which was initially explored, but it did not produce satisfactory results. Therefore, the decision was made that the SVM model and the concepts where only to be used when there was at least one point in each class. The result was that on certain problems, usually only for the first handful of iterations, the concept additions were not used. As this obviously influences the results, the Mimic SVM version was also run on the test problems with the assumption that the concept additions where only added to the base algorithm when there is at least one point in every class.

Here a single model was used to approximate the entire feasibility boundary. However, the use of separate SVM models for each constraint was also considered, as this could lead to more accurate approximations. It was elected to use one model in order to avoid even more additional computational overhead and to avoid complications with regards to approximating the distance to the boundary.

Figure 6.2 visualises the final version of how SVM and PSO interact. The construction of the model of the boundary takes place at the start of every iteration of the PSO unless there is not at least one point in each class. All points that the PSO evaluate are sent to the model as training data. A C-SVM with an RBF kernel was used.

The construction of a sufficiently accurate model through the tuning of the C and γ parameters was automated as shown in the diagram. It is by far the most time-consuming aspect of building the SVM models. Ten-fold cross-validation was used to test the accuracy of the model. The percentage of 95% as the threshold below which tuning occurred was arbitrarily selected based on perceived best performance or what accuracy was perceived to be reasonable to expect from the models.

Some alternative parameter tuning strategies such as a refined grid search and quadratic interpolation were implemented, but ultimately a simple grid search was selected. A 4x4 grid with C values of $[2^{-3}, 2, 2^5, 2^{11}]$ and γ values of $[2^{-11}, 2^{-5}, 2, 2^3]$ was used.

The final CV testing accuracy was recorded. The model was also assessed through its training accuracy, thus how many training points are correctly classified, and, as the constraints for the test problems were available, the true prediction accuracy was calculated.

6.4.2 Utilisation of the SVM's predictions

The information from the SVM models that were used were the class predictions of whether a point would be feasible or infeasible and the predicted

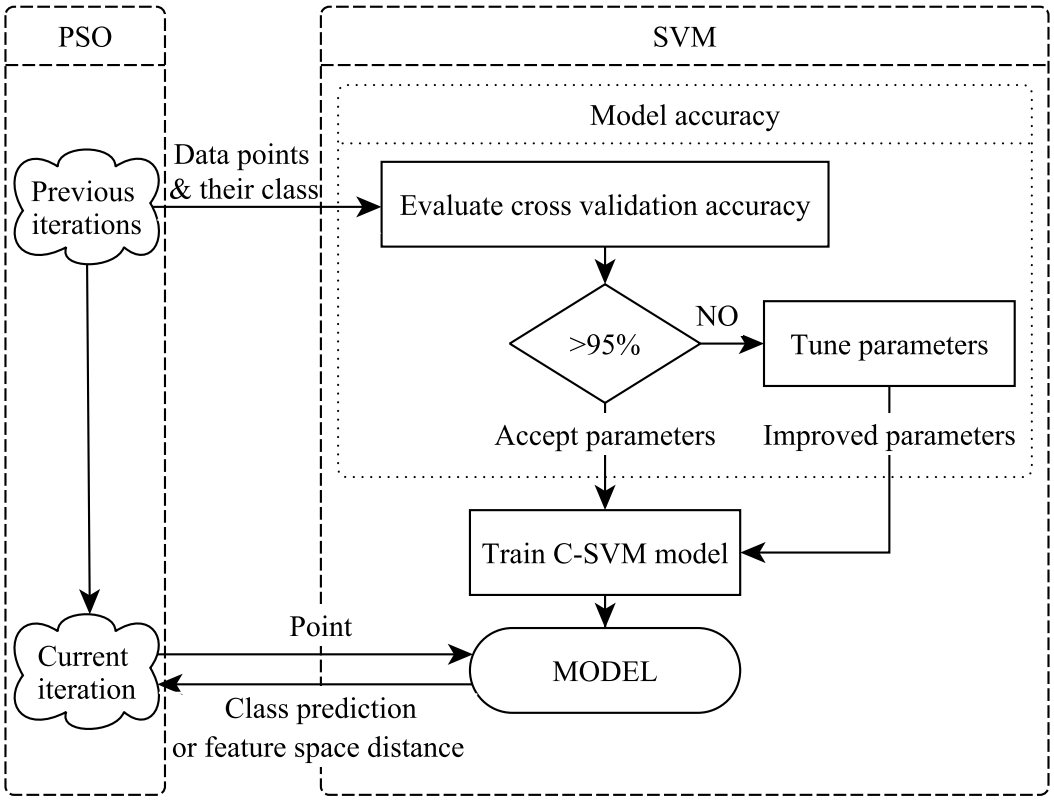

Figure 6.2: SVM and PSO interaction

shortest distance to the boundary in the feature space for a point, depending on the specific concept.

As it was not possible to obtain an inverse map from the feature space, as discussed in section 4.2.4, and so doing obtain the distance in the search space more directly from the model, it had to be approximated. Similar to the Mimic SVM, obtaining the distance was formulated as the solution of a simple optimisation problem where the goal is to minimise the distance between the point of interest and a point on the boundary. Here instead of evaluating the constraint in order to restrict the point to the boundary, the predicted shortest distance to the hyperplane in the feature space was used as a measure of the constraint violation. As there is only one SVM model instead of multiple separate constraints, there is no need for multiple calculations of the shortest distance such as was needed for Mimic SVM. The use of the feature space distance directly in the constraints was also investigated. The solution of this optimisation problem once again also naturally calculates the boundary point that concept 4 requires.

6.4.3 Suggestions for improvement

There are many ways in which the use of the SVM classifier could be improved to reduce the computation time or increase the accuracy of the models. More carefully selecting the data points from the PSO that are sent to the SVM model is one clear area. Especially at higher iterations, there are potentially substantial reductions that can be made to the time the classifier takes to construct a model if the data points were more carefully selected. With the swarm converging on a specific area in the search space, choosing data points from that area and throwing away points that are too far removed could improve the accuracy of the model's predictions for that area. However, one would have to study how different point selection strategies affect performance carefully.

The model could also not be constructed at every iteration. It could be done every fixed number of iterations. Alternatively, one could check whether the model from the previous iteration accurately predicts the class of the new data points and use the model for the new iteration if most are still correctly classified.

There is great potential for reducing the time to tune the SVM model by parallelising the computation of the separate parameter cases when performing a grid search. Also, the number of fold's used for the k-fold cross-validation does have a significant impact on the time it takes to construct a model. Ten folds were selected because some informal testing showed a clear improvement in the quality of models ultimately obtained if the number of folds was increased up to it, but there is some room for varying it if the classifier is taking too long.

In the case of problems with multiple nonlinear constraints, better performance might be achievable through the use of separate SVM models for each constraint, to more accurately approximate them. The extra computational overhead could be reduced by proper parallelisation of the model's construction.

6.5 Testing

6.5.1 Notes on the testing of metaheuristics

The imprecise nature of metaheuristics complicates testing and comparing them to each other well. Their performance is typically highly influenced by how well their parameters are tuned for a specific problem (Hartke, 2011). The stochastic nature of metaheuristics also means that the solution obtained may vary from run to run and thus necessitates comparison over multiple runs on the same problem.

Hooker (1995) strongly argues that there is little value in the competitive testing of one method against another for research purposes. Competitive testing informs one when an algorithm is performing better on a specific problem

or problems, but not why. The problem is how to make a competition fair. The one algorithm might have been carefully tuned, possibly before formal testing, and the test problems selected because the method does well. One competitor's code might be made more efficient and its competitor neglected by comparison. Hooker advocates instead for taking a more scientific approach. Hypotheses should be tested by controlled experiments to study the influence of different features on the performance of the methods. Failures should be communicated. Essentially, rather answer what is improving a heuristics performance and attempt to explain why it is improving performance, than selecting a method that wins some arbitrarily chosen race. He also expressed concern about the selection of the test problems and whether a truly representative testing set is even possible.

Barr, Golden, Kelly, Resende and Stewart (1995) presented advice on how to conduct and report such controlled experiments on heuristics. They state that for heuristics, which are inexact procedures, obtaining an acceptable solution more quickly is more important than the quality of the solution. Also, that heuristics should attempt to be accurate, robust, simple, have a high-impact, generalise well and be innovative. Reports on heuristics must reveal the design procedure, provide insight into the reasons for its performance and behaviour, try to ensure reproducibility and provide statistical analyses of the results. They also describe descriptive experiments that focus on characterising the algorithm, rather than comparing it to others.

More recently, Sörensen (2015) warned of the tendency of metaheuristic research to be pulled in an unscientific direction. He believed that part of the problem is that there is no real widely accepted method about how to test metaheuristic algorithms and that research and testing are often largely unstructured. He called for deconstructing methods in order to test to see why they worked and the production of lean methods with only their essential parts remaining which has clear ties to the structure of the problem that is being solved. Newly proposed metaheuristics should also be properly placed in the frame of the general literature and it must be clearly communicated in what ways an algorithm is conceptually similar to existing methods.

6.5.2 Test problems

The performance of all of the concepts was compared to the base algorithm. This was done as the final goal was to be able to say to other researchers that adding the concepts improves this base, so it would be worth testing it on their PSO version as it could potentially also improve their results. Therefore, it was necessary to test the base PSO's algorithm thoroughly in order to characterise its behaviour.

To characterise an algorithm's behaviour one must ideally evaluate the optimiser's performance on test problems that possess a wide a variety of properties. For this research, in order to be able to draw some conclusions about

how the base reacts to problems possessing certain properties, the test problems were selected in such a way as to attempt to isolate the selected problem properties for comparison. This is not something that is straightforward to accomplish perfectly and must by necessity have limits in its scope to keep everything manageable. However, the greater variety of problems these selection criteria introduces is still of value for providing a better impression of the algorithm's overall capabilities and limitations.

The problem properties were based on Opara and Arabas (2011). The problem properties that the base was tested for and how they were tested for, is as follows.

1. *Large number of design variables*: Compare a low versus a high number of design variables on the same problem.

2. *Linear and nonlinear constraints*: Use the same problem(s), but apply different linear and nonlinear constraints. Alternatively, compare performance on problems with linear versus nonlinear constraints.

3. *Large number of local optima*: Test several problems that have many local optima to see whether the algorithm can still find the global optimum.

4. *Noisiness*: Add noise to (a) problem(s) and evaluate the difference in performance.

5. *Global optimum on the boundary of the feasible space*: Test on problems with this property.

6. *Linearly non-separable functions*: Test on both separable and non-separable functions.

7. *High conditioning of a function*: Test on one or more ill-conditioned test problems (small position change leads to a large change in $f(\boldsymbol{x})$).

8. *Functions with flat areas*: Test on problems with flat or almost flat regions, especially around the global optimum.

Table 6.1 summarises the various test problems that were ultimately selected to provide information on how the algorithm performs on problems that possess the properties mentioned above. All these problems had either a known global solution or a best-found objective function value according to literature, to which results were compared. Most are fairly standard benchmark problems that are commonly used in literature, while others are analytically defined engineering design problems, also from literature. It should be noted that almost all the constrained problems that were selected possess a global optimum located on the feasibility boundary (property 5).

Table 6.2 shows at what stages of the development process each problem was used for testing and what the sources of the problems are. In all testing, 25

runs of an algorithm on a problem were conducted to account for the stochastic nature of PSO.

Table 6.1: Test problem properties

Problem number	n	Constraints			Noise added	Property to test for							
		No. of g	No. of h	Linear/ Nonlinear		1	2	3	4	5	6	7	8
1	5	-	-	-	-					-	s		
1.1	10	-	-	-	-					-	s		
1.2	30	-	-	-	-					-	s		
2	3	-	-	-	No					-	ns		
2.1	3	-	-	-	random					-	ns		
2.2	15	-	-	-	-					-	ns		
3	3	-	-	-	-					-	s		
3.1	15	-	-	-	-					-	s		
4	3	-	-	-	-					-	ns		
4.1	15	-	-	-	-					-	ns		
5	2	-	-	-	-					-	ns		
6	2	-	-	-	No					-	s		
6.1	2	-	-	-	random					-	s		
7	2	-	-	-	-					-	s		
9	2	-	-	-	-					-	ns		
12	4	-	-	-	-					-	ns		
13	2	-	-	-	No					-	ns		
13.1	2	-	-	-	random					-	ns		
16	2	2	-	NL	-						ns		
17	2	2	-	Mixed	-						s		
18	13	9	-	L	-						s		
19*	2	-	1	NL	-						ns		
19.1*	10	-	1	NL	-						ns		
20	6	2	-	L	-						s		
22	4	7	-	Mixed	-						ns		
23	3	4	-	Mixed	-					~	ns		
24	5	6	-	Mixed	-						ns		
25	2	2	-	NL	-						ns		
26	3	1	-	NL	-						s		
29*	5	4	-	L	-					~	s		
30	3	-	1	L	-						ns		
32	13	9	-	L	-						s		
33	20	2	-	NL	-						ns		
34	10	-	1	NL	-						ns		
35	5	6	-	NL	-						ns		
36	4	2	3	Mixed	-						s		
37	2	2	-	NL	-						s		
38	10	8	-	Mixed	-						ns		
39	7	4	-	NL	-						ns		
40	8	6	-	Mixed	-						s		
41	2	-	1	NL	-						s		
42	5	-	3	NL	-						ns		
43	10	-	3	L	-						ns		
44	3	-	2	Mixed	-						ns		
45	6	-	4	NL	-						s		
46	7	1	5	Mixed	-						s		
47	22	1	19	Mixed	-						s		
48	2	2	-	NL	-						s		

$*$ maximisation problem

Note: L = linear, NL = nonlinear, mixed = L & NL, s = separable, ns = non-separable

Table 6.2: Test problems' usage and sources

Problem number	**Stage used in**	**Source**	**Problem name**	**Known solution** $f(\boldsymbol{x}^*)$
1	1	Carlisle and Dozier (2001) and Eberhart and Shi (2000)	Sphere function	0
1.1	1			0
1.2	1			0
2	1		Rosenbrock function	0
2.1	1			0
2.2	1			0
3	1		Generalized Rastrigin function	0
3.1	1			0
4	1		Generalized Griewank function	0
4.1	1			0
5	1		Schaffer's f6 function	0
6	1	Hassan *et al.* (2005)	The Eggcrate function	0
6.1	1			0
7	1	Surjanovic and Bingham (2013)	The Ackley function	$4.44(10^{-16})$
9	1		Dixon-Price function	0
12	1	Engelbrecht (2007)	Colville (p. 25)	0
13	1		Easom (p. 25)	-1
13.1	1			-1
16	All		Problem 1 (p. 33)	0.25
17	All		Problem 2 (p. 34)	1
18	All		Problem 3 (p. 34)	-15
19	All		Problem 4 (p. 34)	1.0 (max)
19.1	All			
20	All		Problem 5 (p. 34)	-213
22	All	Hu *et al.* (2003)	Welded Beam Design	1.7248**
23	All		Weight of a Spring	0.01267**
24	All		Himmelblau's Nonlinear Prob.	-30665.6**
25	All	Vanderplaats (2011)	3-bar truss	2.639**
26	All		Box design	12.0**
29	All		Portfolio Selection	0.298** (max)
30	All		Eq. constraints (p. 93)	$6.7(10^{-5})$**
32	All	Liang *et al.* (2006)	G01	-15
33	All		G02	-0.803619104126**
34	All		G03	-1.00050010001
35	All		G04	-30665.53867
36	All		G05	5126.4967**
37	All		G06	6961.81387558
38	All		G07	25.0725486601
39	All		G09	680.630057374
40	All		G10	7049.24802053
41	All		G11	0.7499
42	All		G13	0.0539415140419
43	All		G14	-47.7648884595**
44	All		G15	961.71502229**
45	All		G17	8853.53967481**
46	All		G21	193.72451007
47	All		G22	236.430975504**
48	All		G24	-5.5080132716

** according to literature

Chapter 7

Findings

This chapter attempts to summarise the most important findings from testing the three stages of the implementation on the selected test problems and from comparing each new stage to the previous. The focus is on answering the most pertinent questions about each stage and highlighting any notable results.

7.1 Base PSO

The purpose of the base PSO algorithm, as detailed in section 6.2, is to provide a reference relative to which the impact of the addition of the concepts can be assessed. Thus, it is advisable to first understand how the base algorithm itself behaves and establish that it is an algorithm that is acceptable for comparison.

The base algorithm was run on the full set of test problems (section 6.5.2), both unconstrained and constrained, for 25 runs of on average 300 iterations, in order to compare it to the later stages. However, the initial assessment of the algorithm's abilities and limits only considered up until problem 30, as the other problems were added later, so those findings are presented here. Problems 32-48 are in any case more complex constrained problem, many of which the base algorithm proved incapable of solving. The results on the unconstrained problems are summarised in table 7.1. The results on the constrained problems are summarised in table 7.2. The default penalty factor of a 1000 was used, except for problem 19.1 where it was 10000 and problem 30 where it was set to 10. σ is the standard deviation in the objective function value of the solutions of the multiple runs. i here represents the number of fixed iterations used. The number of runs that were feasible is reported and for the equality constraints the tolerance with which the best solution from all of the runs complied with the constraint is provided.

The base algorithm worked reasonably well on the more basic constrained and unconstrained problems, with either some increased variability or failure on some of the more difficult problems.

Comparing problem 1 with 1.1 and 1.2, 2 with 2.2, 4 with 4.1 and 19 with

Table 7.1: Base algorithm results on unconstrained problems

Problem number	**Known** $f(\boldsymbol{x}^*)$	i	**Objective function results**		
			f_{avg}	σ	f_{best}
1	0	100	0.0003709	0.0008393	$4.04(10^{-6})$
1.1	0	200	0.0070359	0.007439	0.00018
1.2	0	600	0.85498	0.473477	0.21708
2	0	300	$9.00(10^{-6})$	$4.38(10^{-5})$	$2.23(10^{-14})$
2.1	0	300	0.309265	0.610772	$1.46(10^{-13})$
2.2	0	600	16.971	17.536	1.0004
2.2v2	0	900	14.4333	19.71	0.3643
3	0	300	0.83602	0.60887	$1.85(10^{-13})$
3.1	0	600	20.4415	5.4079	11.9395
4	0	300	0.02567	0.01467	0.00739
4.1	0	300	0.245	0.09465	0.1124
5	0	300	0.001615	0.00355	$2.04(10^{-14})$
6	0	300	$3.13(10^{-13})$	$9.98(10^{-13})$	$8.29(10^{-20})$
6.1	0	300	$9.05(10^{-14})$	$2.30(10^{-13})$	$8.31(10^{-21})$
7	$4.44(10^{-16})$	300	$1.09(10^{-6})$	$3.11(10^{-6})$	$1.29(10^{-9})$
9	0	300	$2.46(10^{-13})$	$9.89(10^{-13})$	0
12	0	400	$3.92(10^{-6})$	$9.89(10^{-6})$	$5.67(10^{-12})$
13	-1	200	-1.0	$1.42(10^{-9})$	-1
13.1	-1	200	-1.25	0.03008	-1.296

Table 7.2: Base algorithm results on constrained problems

Prob. no.	**Known** $f(\boldsymbol{x}^*)$	i	**Objective function results**				**Feasible?**	
			f_{avg}	σ	f_{best}	f_{worst}	**Runs**	**Best?**
16	0.25	200	8.5379	11.0526	0.25	24.2817	25	TRUE
17	1	300	1.0000043	$1.36(10^{-5})$	1	1.0000699	25	TRUE
18	-15	600	-8.843	2.302	-14.779	-6	25	TRUE
19*	1.0	300	0.44119	0.4278	0.99994	0.002	17	$< 10^{-11}$
19.1*		600	0.0112	0.03235	0.1499	0	25	$< 10^{-8}$
20	-213	300	-212.2	1.897	-213	-205	25	TRUE
22	1.7248	300	1.7454	0.02088	1.72499	1.7999	25	TRUE
23	0.01267	300	0.01279	0.000247	0.012665	0.013947	25	TRUE
24	-30665.6	300	-31025.5	0.12413	-31025.6	-31025.1	25	TRUE
25	2.639	300	2.63896	$1.15(10^{-6})$	2.638958	2.638963	25	TRUE
26	12	300	12.0011	0.0021	12.00001	12.0105	25	TRUE
29*	0.298	300	0.3643	0.00459	0.37	0.3572	25	TRUE
30	$6.7(10^{-5})$	300	0.015167	0.03977	$2.75(10^{-5})$	0.1998766	17	$< 10^{-11}$

$*$ maximisation problem

19.1 there is a clear decrease in accuracy and precision with an increase in the number of design variables. In the case of problems 3.1 (fifteen variables, Generalized Rastrigin) and 19.1 (ten variables, one equality constraint), the algorithm's best optimum is nowhere near the best available global optimum.

Looking at an equality constraint, for the simple cases that were considered the algorithm managed to find the optimum with reasonable accuracy for two or three design variables after a sufficient number of iterations, but not with much consistency. With the ten variables in problem 19.1 it is clear that the algorithm struggles with equality constrained problems. No amount of tuning of the penalty factor changed this.

These cases are to be expected as it is known that a PSO algorithm that uses a simple penalty method may struggle with equality constraints, and global optimization techniques are typically only used on problems with a smaller number of design variables.

Problem 3.1 appears to have a issue with the swarm occasionally converging onto one of the problem's numerous local minima (that are fairly steep), this could be addressed by further tuning of the algorithm parameters to encourage more exploration by the swarm, maybe by using a larger starting velocity or increasing the craziness. The other problems still manage to find their respective optima to reasonable accuracy in the best case, if not that reliably.

There does not appear to be a clear difference in the algorithm's ability to solve linear, nonlinear or mixed constrained problems. The algorithm somewhat managed both the single linear and nonlinear equality constraints for two or three variables in problems 30 and 19.

Problems 3-7 all possess many local minima (although they differ in nature). The algorithm, as per its nature as a global optimisation method, manages to find the global optimum in each of these problems, except for problem 3.1 with its increased number of design variables.

Problems 2.1, 6.1 and 13.1 had random noise added (increase/decrease factor sampled from a normal distribution). Problems 2.1 and 6.1 show no notable difference in the best solution found from problems 2 and 6. However, the spread in the solutions obtained for 2.1 noticeably increased from problem 2. This makes sense as the overall 'randomness' of the problem has increased with the addition of noise. Problem 13.1 has solutions that are clearly below the global optimum of the problem, the error being caused by the addition of noise.

The problems are a mixture of both linearly separable and non-separable functions, thus the algorithm appears at least mostly capable of solving both types. There are obviously many other factors at work influencing the performance of the algorithm on a problem relative to another thus it is difficult to see any noticeable pattern of difficulty with solving non-separable problems.

Problem 5 (Schaffer's f6 function) has very deep thin folds surrounding its optimum, but the algorithm appears to manages this without a problem.

Problem 13 (Easom) has a very steep valley around the optimum and the algorithm also had no issue in finding it.

7.2 Mimic SVM

The Mimic SVM stage, as detailed in section 6.3, added each concept onto the base algorithm while simulating the predictions that they require. The goal was to see the full effect each idea has the potential to have on the performance of the base, as the 'SVM model' is effectively 100% accurate. Not having the SVM models attached also keeps the computation time on the problems more manageable, allowing for more runs to be executed on the numerous test problems and thus the full ability of the concepts to be revealed.

All four concepts, thus nine concept variations, for Mimic SVM was run on all of the constrained problems from section 6.5.2, thus 30 problems, for 25 runs of on average 300 iterations. Only some of the more notable results are summarised here.

In general, each of the concepts caused at least some form of improvement relative to the base on some of the problems. There were for some concepts significant reductions in the number of iterations, improvements in the consistency with which a good optimum is located between different runs and remarkably clear improvements on some simple equality constrained problems. There were even slight improvements for some of the problems that the base algorithm already performed well on. However, neither the base nor the concepts could solve problems 40-47, which all possess more complex feasible boundaries consisting of mostly multiple equality constraints with tiny feasible regions. The concepts thus did provide at least some improvement upon problems that had primarily inequality constraints with optima on the boundary and simple equality constraints.

C1o4 and C1o5 were generally less time consuming to run, as they only use class predictions. While C2 and C1o2 where generally by far the most time consuming to run as they require numerous calculations of the shortest distance to the boundary.

The variations of concepts 1 were the most reliable in terms of producing an improvement in the solutions. Concept 2 occasionally degrades the quality of the solutions.

Figures 7.1 - 7.6 shows some results obtained for concept 1's variations. For concepts 2, C3o1, C3o2 and 4 figures 7.7 and 7.9 showcase some of their performance. Figures 7.10 and 7.11 show some combined results for all the concepts. Figure 7.12 shows some results of C1o4, C1o5 and C4. These figures show some of what has been described above. Note again that problems 19, 19.1 and 29 are maximisation problems and that problems 19, 19.1, 30, 34 and 36 have equality constraints.

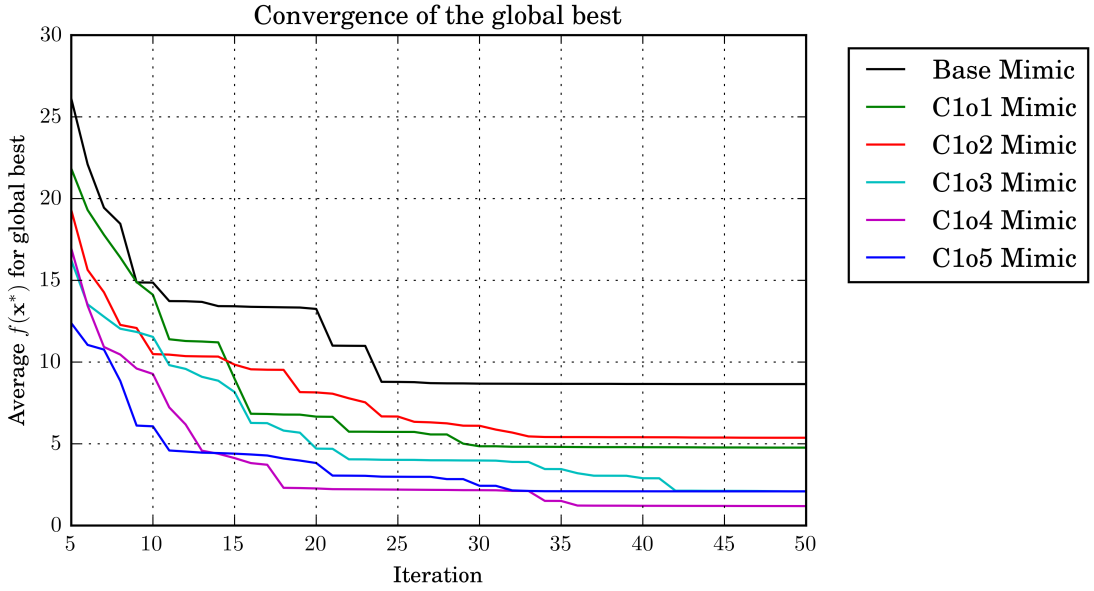

Figure 7.1: Mimic SVM's concept 1 vs base on problem 16

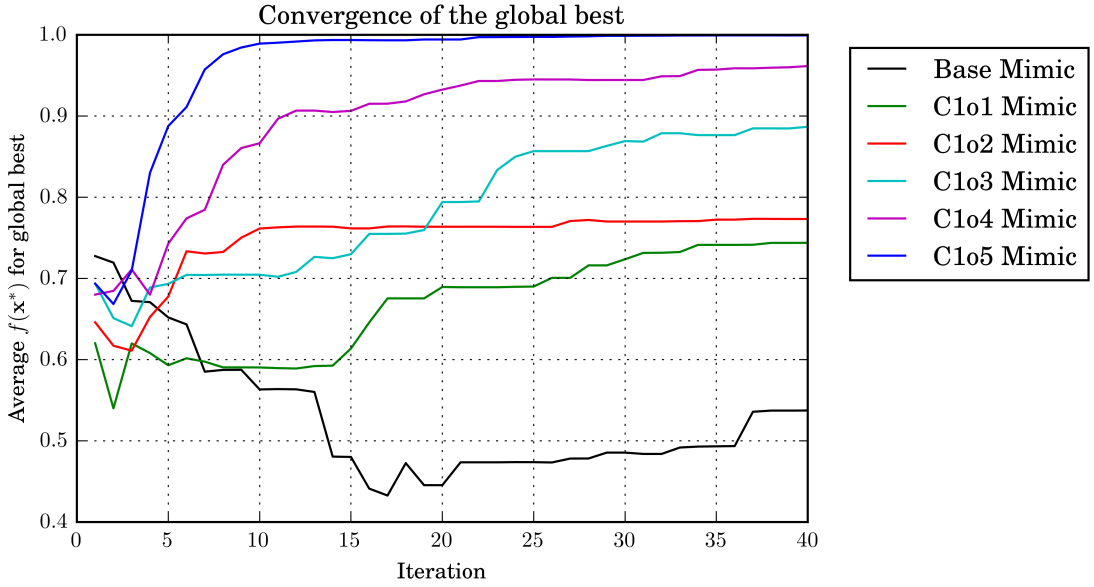

Figure 7.2: Mimic SVM's concept 1 vs base on problem 19

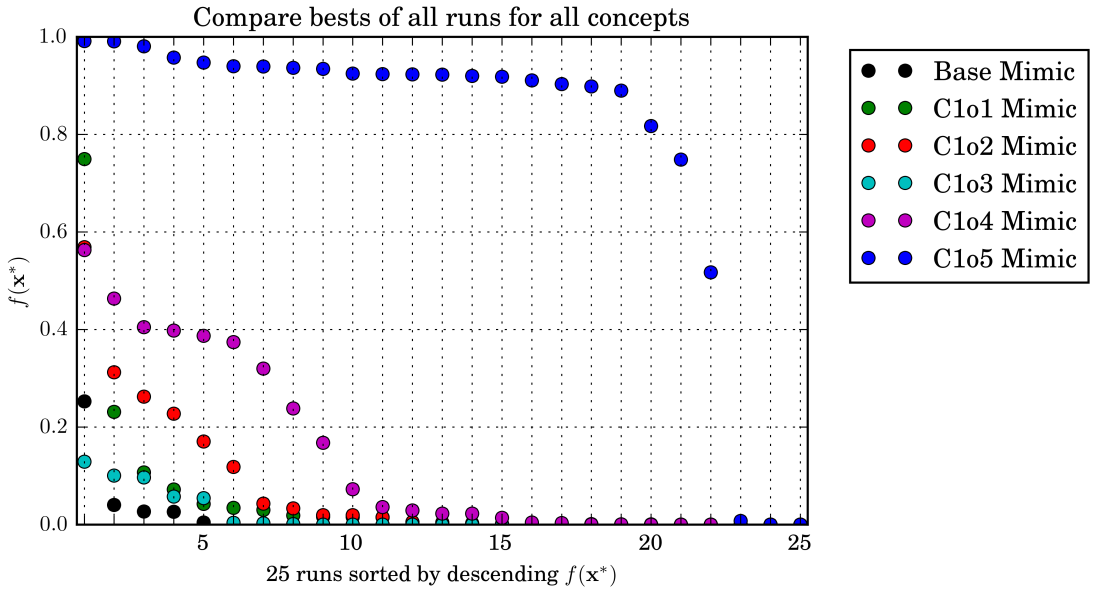

Figure 7.3: Mimic SVM's concept 1 vs base on problem 19.1

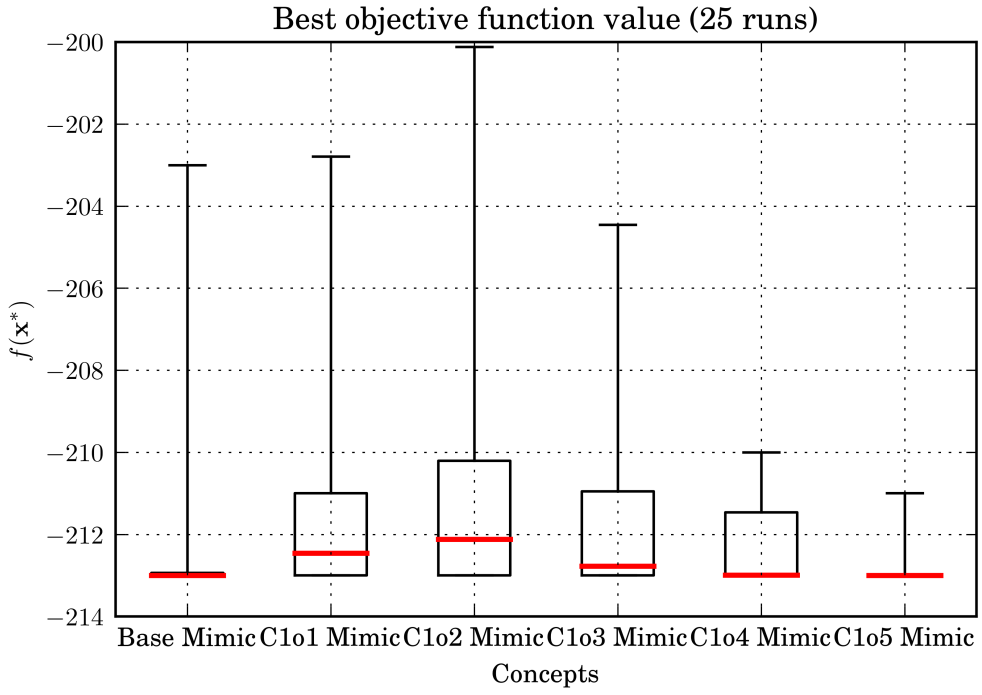

Figure 7.4: Mimic SVM's concept 1 vs base on problem 20

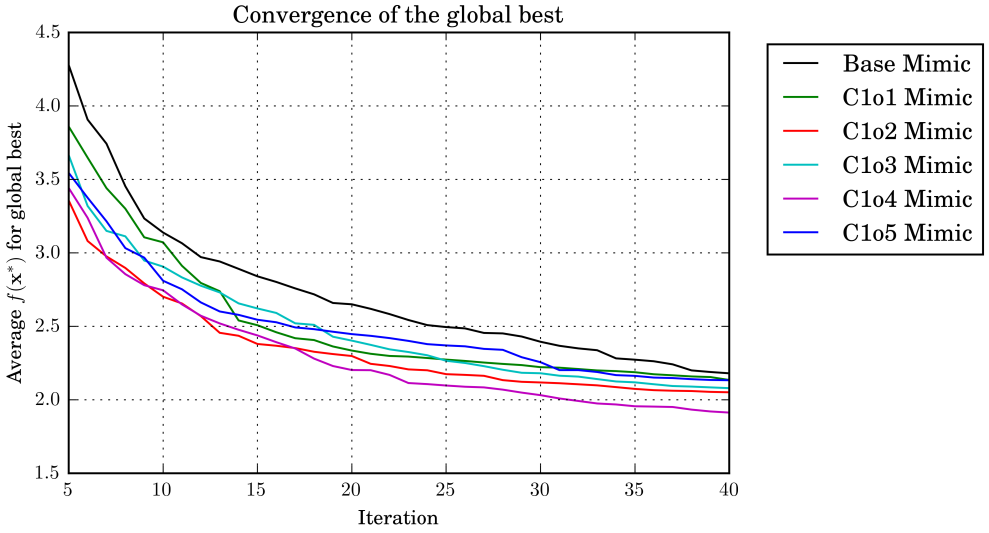

Figure 7.5: Mimic SVM's concept 1 vs base on problem 22

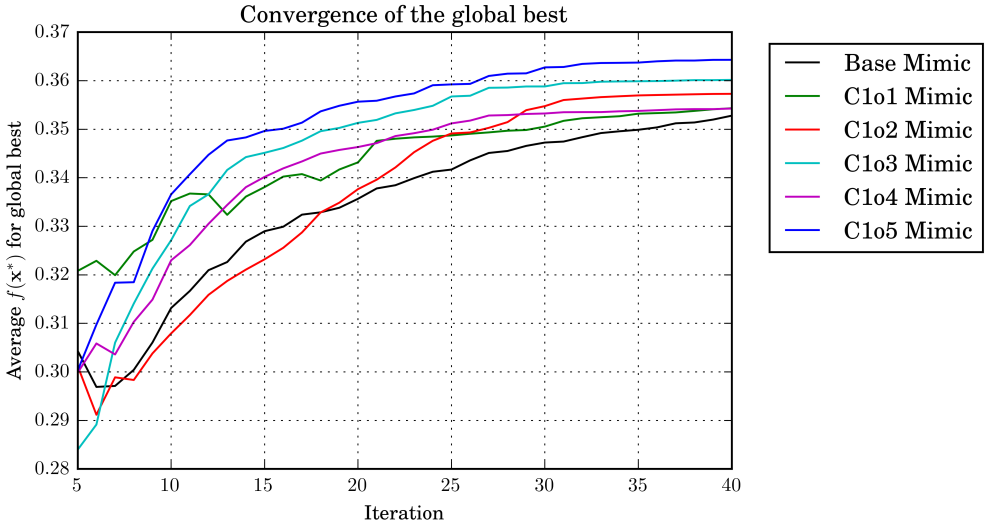

Figure 7.6: Mimic SVM's concept 1 vs base on problem 29

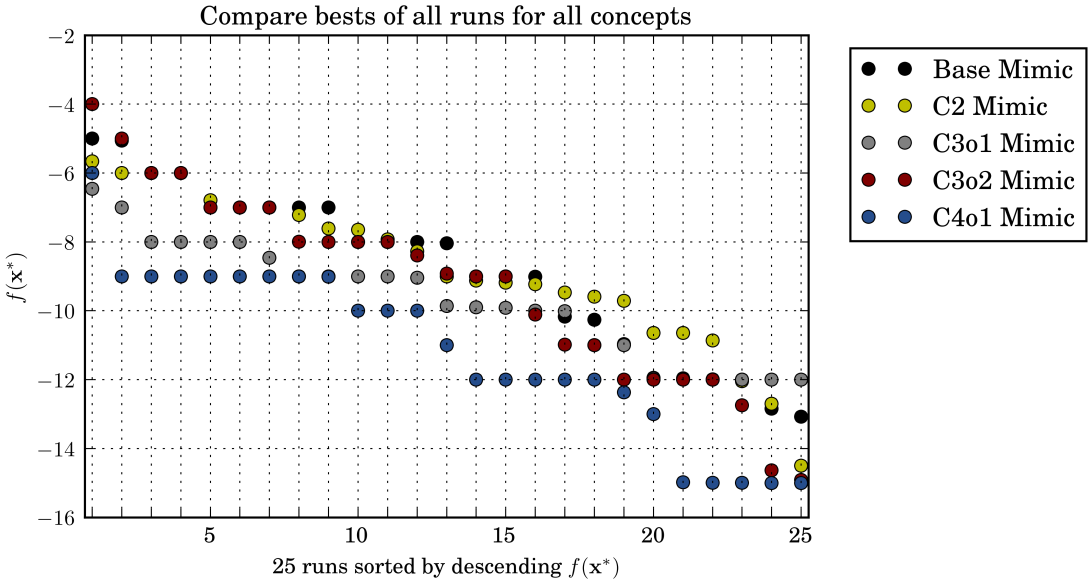

Figure 7.7: Mimic SVM's concepts 2, 3, 4 vs base on problem 18

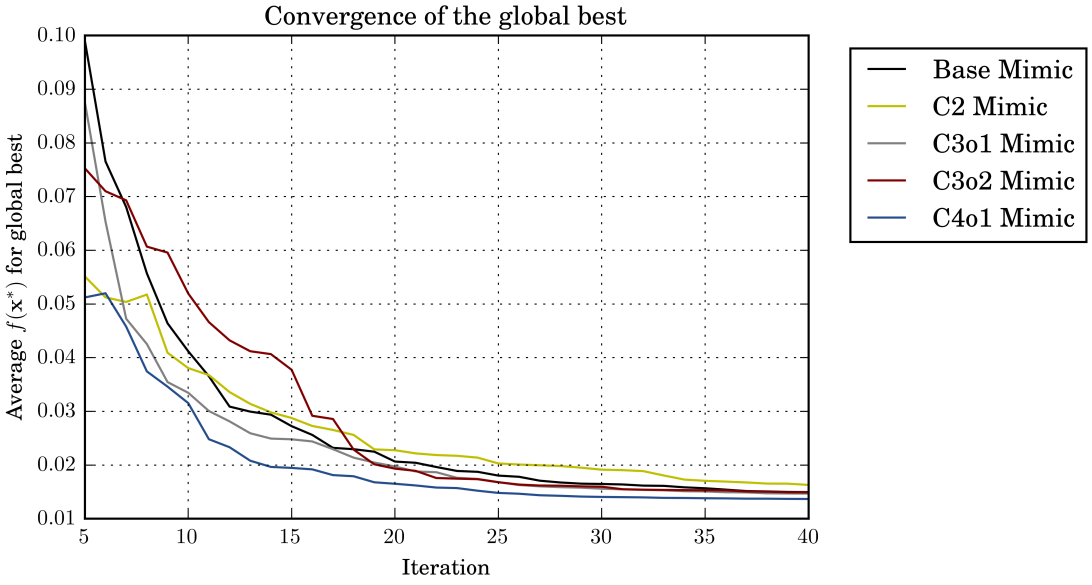

Figure 7.8: Mimic SVM's concepts 2, 3, 4 vs base on problem 23

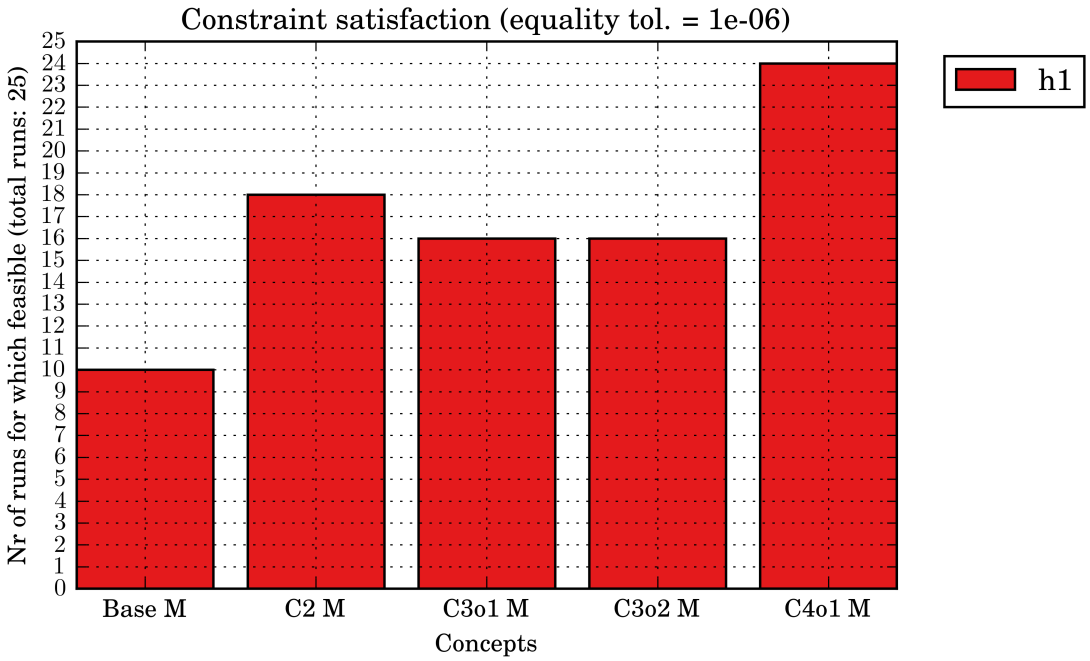

Figure 7.9: Mimic SVM's concepts 2, 3, 4 vs base on problem 30

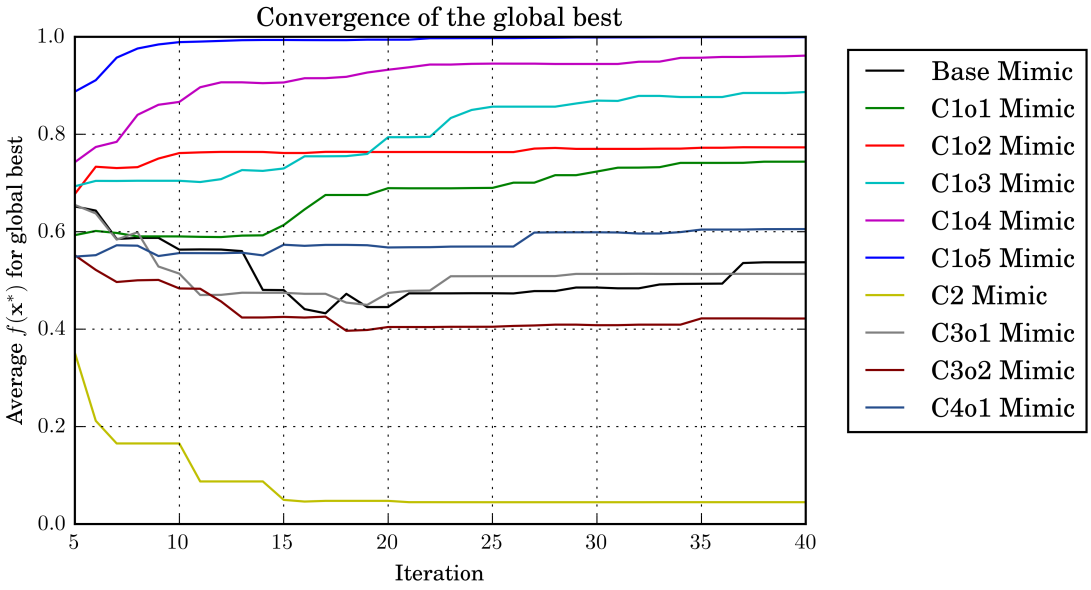

Figure 7.10: Mimic SVM's all concepts vs base on problem 19

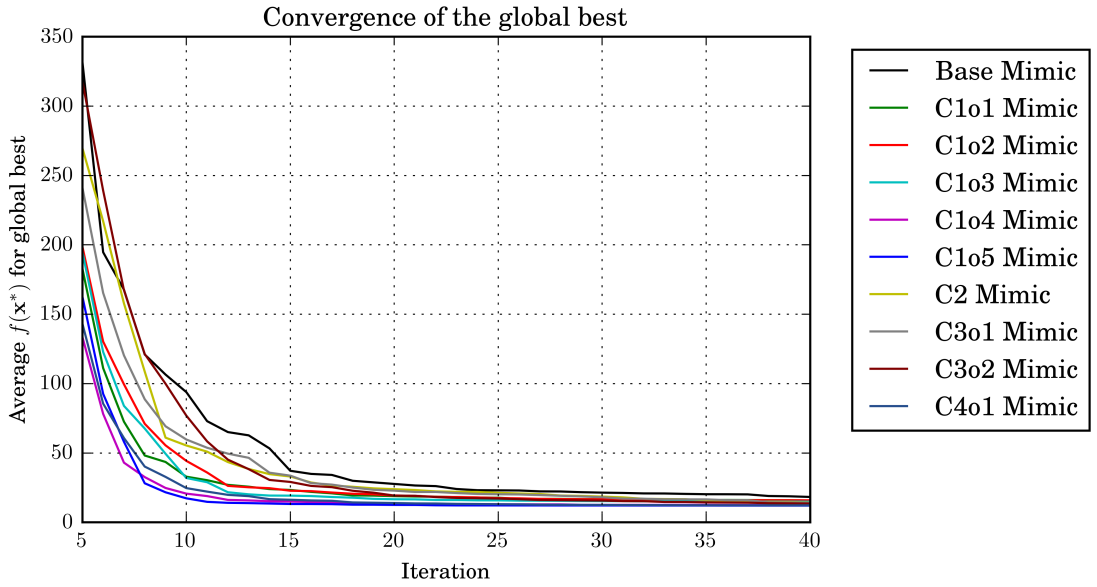

Figure 7.11: Mimic SVM's all concepts vs base on problem 26

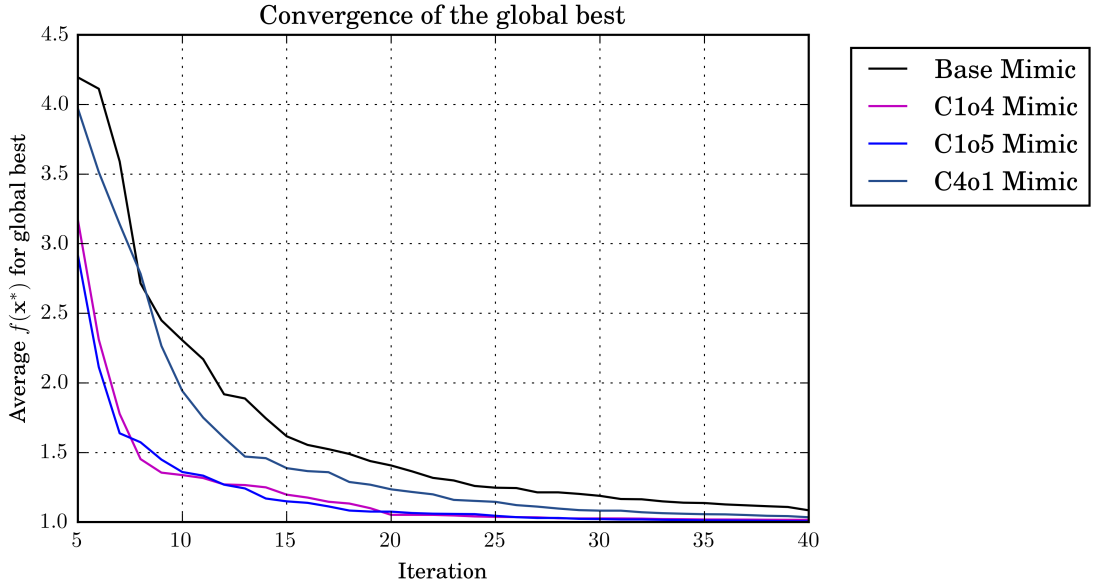

Figure 7.12: Mimic SVM's C1o4, C1o5, C4 vs base on problem 17

7.3 Full SVM

The full SVM stage replaced the simulated predictions of stage 2 with the actual full SVM implementation. By comparing it to the Mimic SVM stage one can get a sense of the impact the addition of the SVM has with regards to the quality of the solutions that the concept variations provide, the additional computational overhead or computation time it introduces and the complexity it introduces to the algorithm.

The nine concept variations were run on all 30 constrained problems from section 6.5.2 for 25 runs of on average 40 iterations. This was done for both the standard uniform random inisialisation and Latin hypercube sampling. The concept variations that use the distance were also rerun using the feature space distance directly (random inisialisation). The Mimic SVM version of the concepts as well as the base algorithm were also rerun for 40 iterations to allow for comparison.

The reduced number of iterations executed is a direct reflection of the increased computation time introduced by the models, as even with the reduced number of iterations the computations take far longer. However, on most of the concepts these first few iterations are where the largest changes in the best-found solution occurs and thus they do provide a sufficient look at the impact that adding the SVM classifier has on the results.

In general, the concepts performed slightly worse in the Full SVM version. C1o5 and C1o4 that only use class predictions were less affected. The impression is that more calculations of the distance tend to negatively impact the performance.

On the problems 40-47 that the algorithms failed on, which have very tiny feasible regions ($F \ll 1\%$ of S), the classifier could also not manage to construct a model as it experienced difficulty with finding at least one point in the feasible region.

The concepts that used the shortest distance to the boundary were also run while using the feature space distance directly. This did not work well and generally degraded the results.

The impact of using Latin hypercube sampling for initialisation varied from problem to problem and from concept to concept, sometimes improving the results, sometimes degrading it slightly. Further investigation is needed to see whether there is a pattern to this variation.

On average, the training and testing accuracy for problems were similar, but tended to degrade slightly over the course of a run. The prediction accuracy was occasionally significantly lower than these, especially at the start of a run and then peaked at some value over the course of the run.

Figures 7.13 - 7.20 provides some results for the concepts with the full SVM attached. They show some of what has been described above.

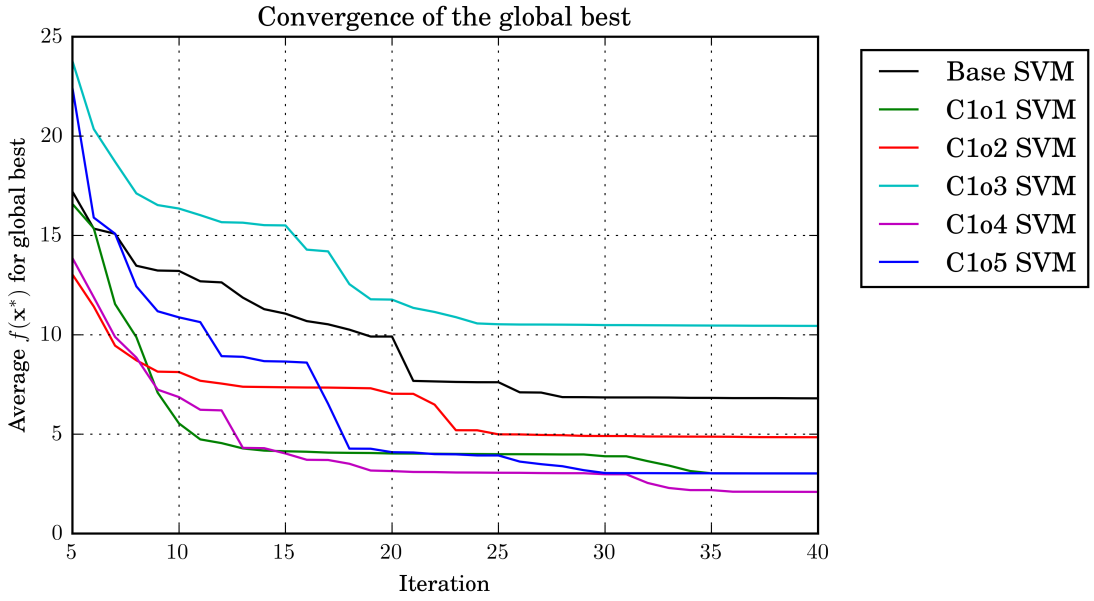

Figure 7.13: Full SVM's concept 1 vs base on problem 16

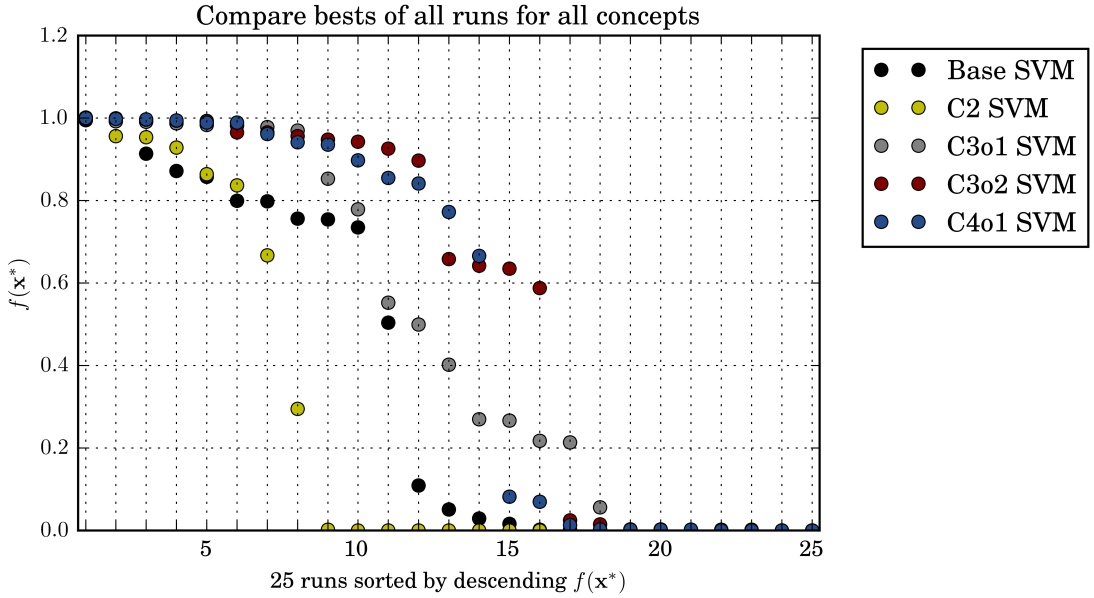

Figure 7.14: Full SVM's concepts 2, 3, 4 vs base on problem 19

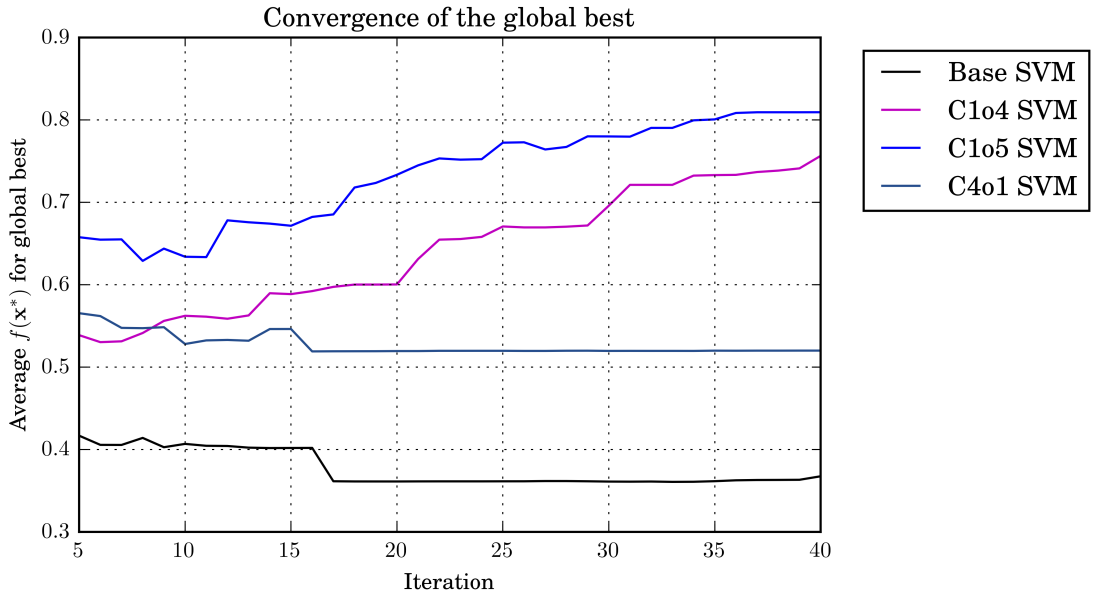

Figure 7.15: Full SVM's C1o4, C1o5, C4 vs base on problem 19

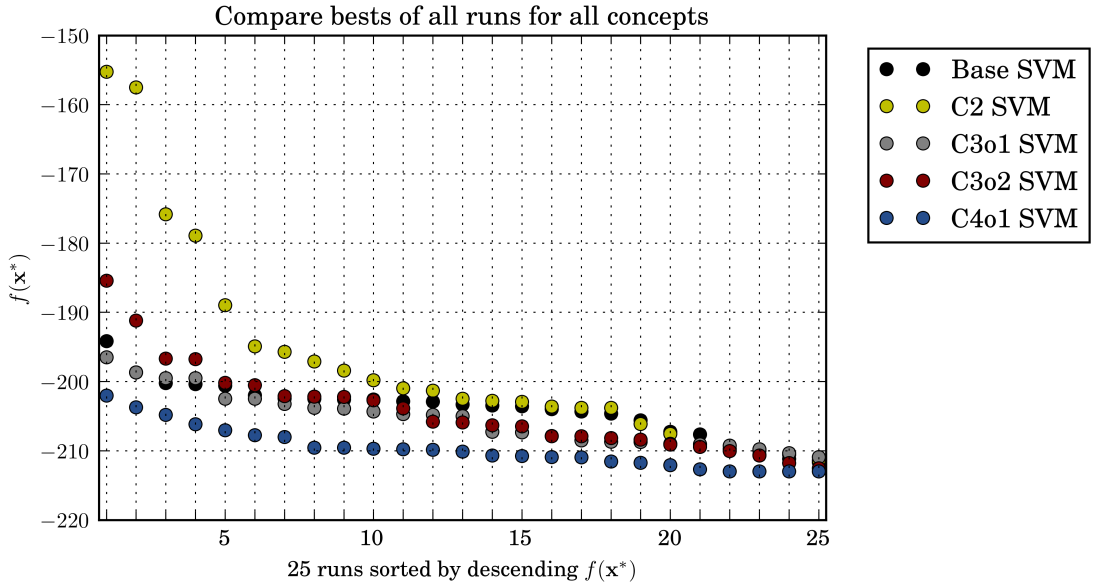

Figure 7.16: Full SVM's C1o4, C1o5, C4 vs base on problem 20

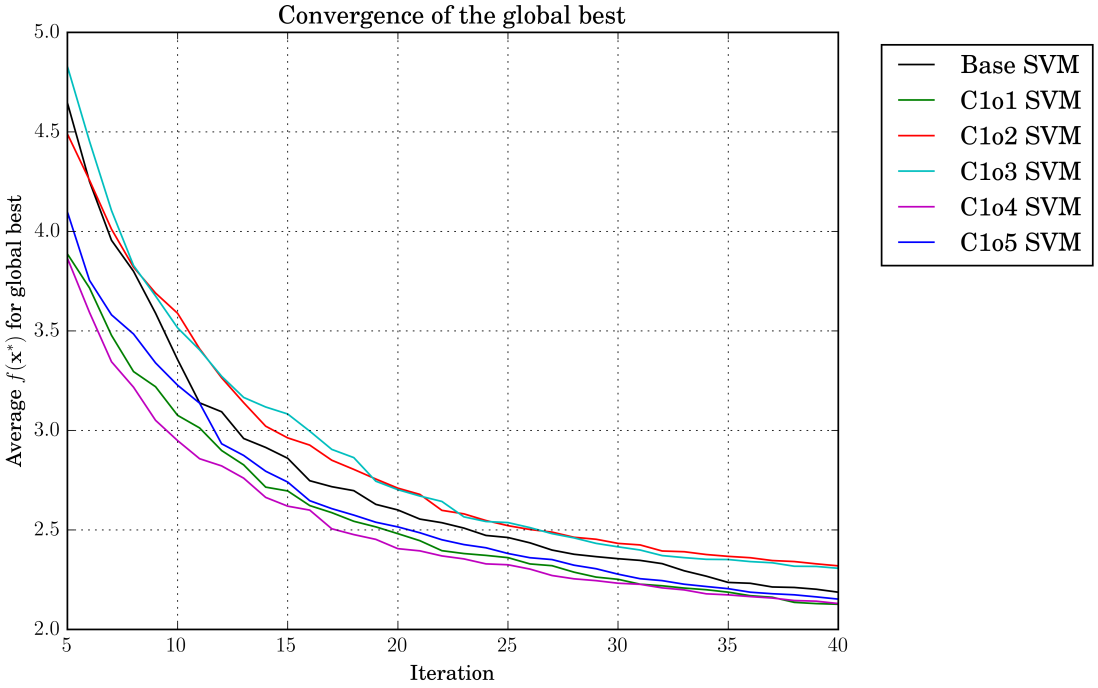

Figure 7.17: Full SVM's concept 1 vs base on problem 22

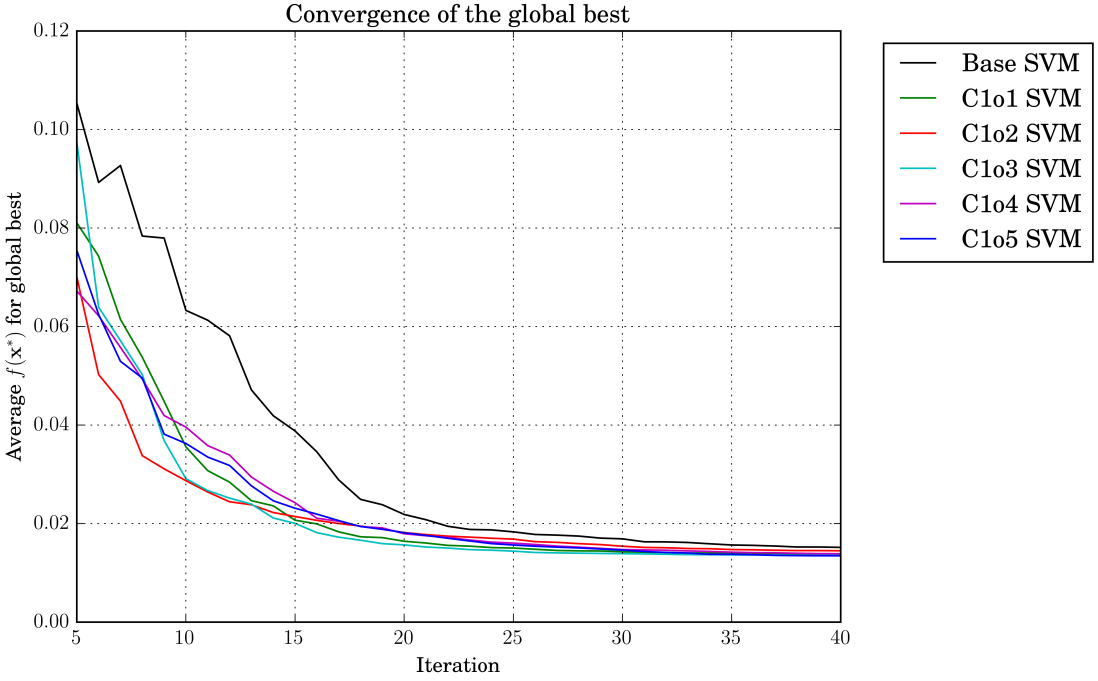

Figure 7.18: Full SVM's concept 1 vs base on problem 23

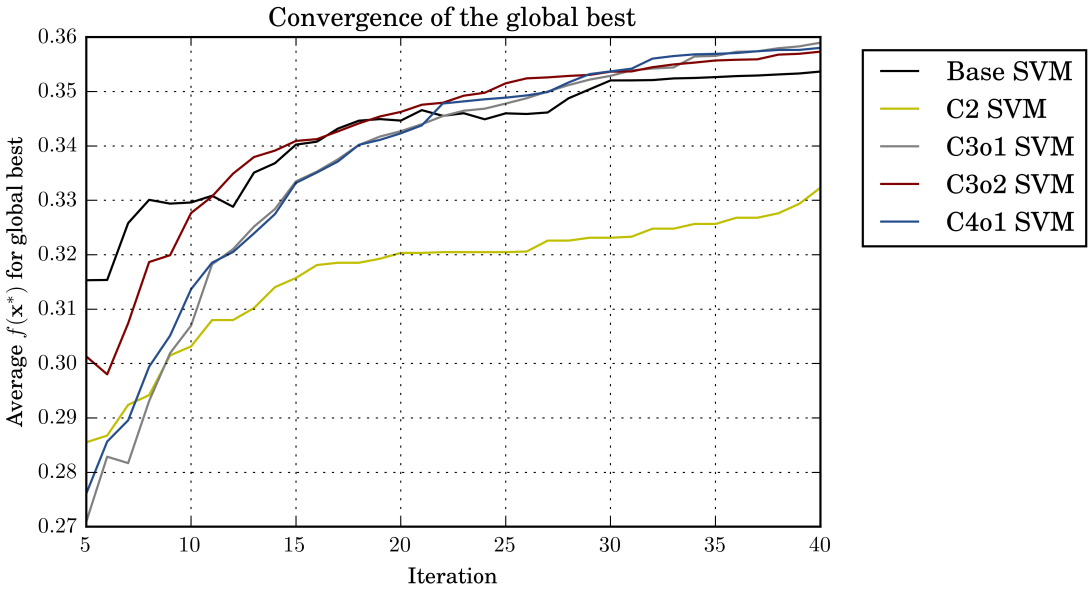

Figure 7.19: Full SVM's concepts 2, 3, 4 vs base on problem 29

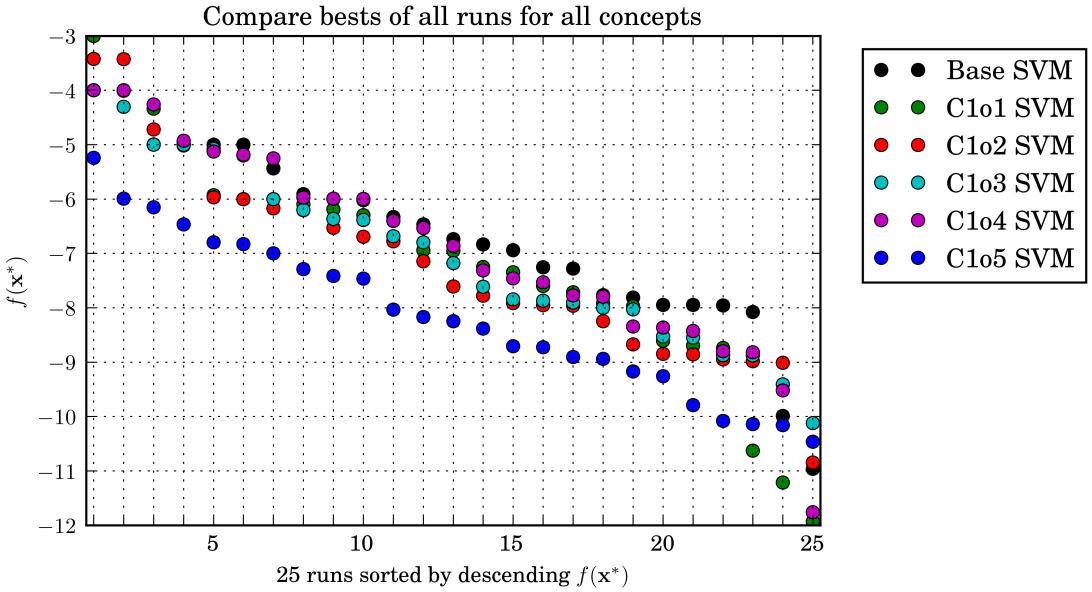

Figure 7.20: Full SVM's concept 1 vs base on problem 32

Chapter 8

Conclusion

8.1 General overview

PSO, along with other similar metaheuristics, may struggle when solving constrained optimisation problems where the global optimum is located on the boundary of the feasible region, as they typically are for most real-world problems, and when there are equality constraints. This is especially of concern to optimisation problems encountered in engineering and more generally when the function evaluations for a problem are expensive.

This thesis has given an overview of the development of new methods to adapt PSO to solve these types of problems. The new methods all use support vector classification models to approximate the boundary of the feasible region by using the data gathered by the swarm. This newly extracted information is used to encourage the swarm to explore the boundary in various ways.

A thorough literature review was presented that places the concepts into their proper context in literature, provides the necessary background knowledge for understanding the work, explains related or similar research, highlights problems or challenges and motivates the direction this research took.

The reasoning behind how the concepts were created was laid out. Four concepts, nine variations in total, were designed for testing. How they work, any similarities they may bear to existing ideas or methods, possible pitfalls they may encounter and possible variations or improvements were all detailed.

The concepts were implemented in three main stages to allow for more easy development and the comparison of the effect of different factors. This was detailed. The selected test problems were also presented and motivated.

The stages were all evaluated through testing on the large set of test problems. The concepts were considered to be additions to a baseline PSO algorithm, and the impact their addition had was evaluated relative to it. The concepts were both assessed while mimicking the SVC's predictions, thus assuming 100% model accuracy, and when the full SVM classifier was attached.

Overall, several of the concepts provided significant reductions in the num-

ber of iterations that were required to achieve similar objective function values on many of the test problems. Often accompanied by decreased variation in the results obtained from different runs. There were also clear improvements on some of the simpler equality constrained problems. All of these improvements are especially clear when the predictions of the SVM were only simulated in the Mimic SVM stage. Neither the concepts nor the base could solve some of the highly constrained problems that typically had multiple equality constraints and very tiny feasible spaces.

Adding the SVM classifier did reduce the extent of improvements observed, as initially expected, although there were still clear improvements relative to the base algorithm. The SVM's addition also significantly increased the computation time. Occasionally on some of the test problems with especially small feasible regions, the program could not run the classifier at all, as the algorithm could not find at least one point in the feasible region to provide.

Problems and challenges that were encountered were explained throughout the thesis. Suggestions were made for improving any future implementations.

8.2 Final recommended concept

The concept that is recommend for use is concept 1. Concept 1 shifts particles for which the position rule's new position would entail a move from feasible to infeasible region, back to the approximate point the particle would have to cross the feasibility boundary. Its option 5 (C1o5) that uses a version of the Golden Section method which was modified to use class predictions to perform the line search, should be the first choice.

Many of the other concept variations did outperform it for certain metrics on certain problems and they could possibly be improved. However, it more consistently showed improvements in the solutions, reductions in the number of function evaluations and reductions in the variation of results between runs on many of the problems. Using option 5 and possibly option 4 for the line search is also best, as solely using class predictions reduces the complexity associated with implementing the concepts and the computation time that is required. The use of class predictions also allows for easily exchanging the SVM for another classifier. Additionally, options 5 and 4 also showed the most improvement in the solutions obtained and the most consistent improvement between runs for some of the equality constrained problems.

8.3 Recommendations for future research

Any future research should, first and foremost, focus on improving the efficiency of the creation of the support vector classification models in order to

reduce the computation time, as per the suggestions provided. It could also be worth exploring using other machine learning classification techniques.

The concepts should be tested and used as additions to more efficient or well-established base PSO algorithms. The potential variations on and improvements to the concept algorithms that were suggested could be explored further, as some of the concepts do warrant a second look, such as concept 4 that occasionally performed very well.

Many other concepts could be generated based on the principles that were followed for creating the concepts. The reasoning and development process that was used was laid out in detail in this work, specifically in order to facilitate and encourage this.

The recommended concept should be tested on constrained engineering design optimisation problems that have function evaluations performed by, for example, a finite element analysis or computational fluid dynamics simulation. Also, the concept should be tested on constrained problems that do not have an optimum on the boundary, to verify that it does not hinder the solution of such problems.

A statistical analysis should be conducted of the impact of changing the concepts' parameters on the search. The impact of adding the concepts on the rotation invariance of the linear PSO should also be investigated.

This work is, in essence, a first step. Many of the concepts that were tested obtained definite improvements in their results relative to the base PSO algorithm when solving test problems with a global optimum located on the feasibility boundary or even simpler equality constraints. Therefore, this work shows that there are benefits to be gained from improved knowledge of the location of the feasibility boundary through machine learning based modelling and the addition of mechanisms to PSO algorithms that encourage its exploration. Thus there should be more research conducted to explore the use of these ideas further.

References

Arora, J.S. (2004). *Introduction to Optimum Design*. 2nd edn. Elsevier Academic Press, USA. ISBN 0-12-064155-0.

Aziz, N.A.A., Mohemmed, A.W., Alias, M.Y. and Aziz, K.A. (2011). Particle swarm optimization for constrained and multiobjective problems: A brief review. *IPEDR*, vol. 6, pp. 146–150. ISSN 2010-4626. IACSIT Press, Bali, Indonesia. Available at: `http://www.ipedr.com/vol6/29-A10025.pdf`

Banks, A., Vincent, J. and Anyakoha, C. (2007). A review of particle swarm optimization. Part I: Background and development. *Natural Computing*, vol. 6, no. 4, pp. 467–484. ISSN 1567-7818.

Banks, A., Vincent, J. and Anyakoha, C. (2008). A review of particle swarm optimization. Part II: Hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing*, vol. 7, no. 1, pp. 109–124. ISSN 1567-7818.

Bansal, J.C., Singh, P.K., Saraswat, M., Verma, A., Jadon, S.S. and Abraham, A. (2011). Inertia weight strategies in particle swarm optimization. In: *2011 Third World Congress on Nature and Biologically Inspired Computing*, pp. 633–640.

Barr, R.S., Golden, B.L., Kelly, J.P., Resende, M.G.C. and Stewart, W.R.J. (1995). Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, vol. 1, pp. 9–32.

Basudhar, A., Dribusch, C., Lacaze, S. and Missoum, S. (2012 Aug). Constrained efficient global optimization with support vector machines. *Structural and Multidisciplinary Optimization*, vol. 46, no. 2, pp. 201–221. ISSN 1615-1488.

Beheshti, Z. and Shamsuddin, S.M.H. (2013 January). A review of population-based meta-heuristic algorithm. *International Journal of Advances in Soft Computing and its Applications*. ISSN 2074-8523.

Bonyadi, M.R. and Michalewicz, Z. (2015). Locating potentially disjoint feasible regions of a search space with a particle swarm optimizer. In: Datta, R. and Deb, K. (eds.), *Evolutionary Constrained Optimization*, Infosys Science Foundation Series, pp. 205–230. Springer, New Delhi. ISBN 978-81-322-2184-5.

Bonyadi, M.R. and Michalewicz, Z. (2017 March). Particle swarm optimization for single objective continuous space problems: A Review. *Evolutionary Computation*, vol. 25, no. 1, pp. 1–54. ISSN 1063-6560. arXiv:1401.1942v1.
Available at: http://www.mitpressjournals.org/doi/10.1162/EVCO_r_00180

Bonyadi, M.R., Michalewicz, Z. and Li, X. (2014). An analysis of the velocity updating rule of the particle swarm optimization algorithm. *Journal of Heuristics*, vol. 20, no. 4, pp. 417–452. ISSN 1572-9397.

Boussaïd, I., Lepagnot, J. and Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, vol. 237, pp. 82–117. ISSN 0020-0255.

Carlisle, A. and Dozier, G. (2001). An off-the-shelf pso. *Proceedings of the Workshop on Particle Swarm Optimization 2001.*

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G.A., Burgard, W., Kavraki, L.E. and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, USA. ISBN 9780262255912.

Clerc, M. (1999). The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 1999*, pp. 1951–1957.

Clerc, M. (2006). *Particle Swarm Optimization.* Wiley-ISTE. ISBN 978-1-905209-04-0.

Cormen, T.H., Leiserson, C.E., Rivest, R. and Stein, C. (2009). *Introduction to Algorithms.* Third edition edn. MIT Press, United States of America. ISBN 978-0-262-53305-8.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and other Kernel-based Learning Methods.* Cambridge University Press, Cambridge, UK. ISBN 0-521-78019-5.

Daume III, H. (2004 March). From zero to reproducing kernel hilbert spaces in twelve pages or less.
Available at: http://www.umiacs.umd.edu/ hal/docs/daume04rkhs.pdf

De Castro, L.N. (2007). Fundamentals of natural computing: An overview. *Physics of Life Reviews*, vol. 4, no. 1, pp. 1–36. ISSN 1571-0645.

Demidova, L., Nikulchev, E. and Sokolova, Y. (2016). The SVM classifier based on the modified particle swarm optimization. *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 2, pp. 16–24. ISSN 2156-5570. 1603.08296.

Di, W. (2011). LIBSVM 3.0 extended.
Available at: `https://github.com/VanessaD/libsvm-3.0.extended.wdi`

Eberhart, R.C. and Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2000*, pp. 84–88.

Eiben, A.E. and Smith, J.E. (2007). *Introduction to Evolutionary Computing*. Natural Computing Series. Springer-Verlag Berlin Heidelberg. ISBN 978-3-540-40184-1.

Engelbrecht, A.P. (2005). *Computational Intelligence: An Introduction*. John Wiley & Sons, Great Britain. ISBN 0-470-84870-7.

Engelbrecht, A.P. (2007). *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons Ltd., Great Britain. ISBN 978-0-470-09191-3.

Fehr, J., Arreola, K.Z. and Burkhardt, H. (2008). Fast support vector machine classification of very large datasets. In: Preisach, C., Burkhardt, H., Schmidt-Thieme, L. and Decker, R. (eds.), *Data Analysis, Machine Learning and Applications*, pp. 11–18. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-78246-9.

Feinberg, J. and Langtangen, H.P. (2015). Chaospy: An open source tool for designing methods of uncertainty quantification. *Journal of Computational Science*, vol. 11, pp. 46–57. ISSN 1877-7503.

Forsyth, R. and Rada, R. (1986). *Machine learning applications in expert systems and information retrieval*. Ellis Horwood Series in Artificial Intelligence. Ellis Horwood Limited, Chichester. ISBN 0-85312-947-9.

Gao, G., Sun, C., Zeng, J.-C. and Xue, S. (2015). A constraint approximation assisted PSO for computationally expensive constrained problems. *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, pp. 1354–1359.

Haftka, R.T., Villanueva, D. and Chaudhuri, A. (2016). Parallel surrogate-assisted global optimization with expensive functions - a survey. *Structural and Multidisciplinary Optimization*, vol. 54, no. 1, pp. 3–13. ISSN 1615-1488.

Handoko, S.D., Keong, K.C. and Soon, O.Y. (2008). Using classification for constrained memetic algorithm: A new paradigm. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pp. 547–552. ISSN 1062-922X.

Handoko, S.D., Keong, K.C. and Soon, O.Y. (2009). Classification-assisted memetic algorithms for equality-constrained optimization problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5866 LNAI, no. 2, pp. 391–400. ISSN 0302-9743.

Handoko, S.D., Keong, K.C., Soon, O.Y. and Chan, J. (2011). Classification-assisted memetic algorithms for solving optimization problems with restricted equality constraint function mapping. *2011 IEEE Congress of Evolutionary Computation, CEC 2011*, pp. 1209–1216.

Hartke, B. (2011). Global optimization. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 1, no. 6, pp. 879–887. https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcms.70.
Available at: https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.70

Hassan, R., Cohanim, B., de Weck, O. and Venter, G. (2005). A comparison of particle swarm optimization and the genetic algorithm. *46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, pp. 1–13. Austin, Texas.
Available at: http://arc.aiaa.org/doi/pdf/10.2514/6.2005-1897

Helwig, S., Branke, J. and Mostaghim, S. (2013 April). Experimental analysis of bound handling techniques in particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 2, pp. 259–271. ISSN 1089-778X.

Herbrich, R. and Bach, F. (2001). *Learning Kernel Classifiers: Theory and Algorithms*. Adaptive Computation and Machine Learning series. MIT Press. ISBN 9780262263047.
Available at: https://books.google.co.za/books?id=el_cDjHdP0cC

Hooker, J.N. (1995). Testing heuristics: We have it all wrong. *Journal of Heuristics*, vol. 1, no. 1, pp. 33–42.

Hsu, C.-W., Chang, C.-C. and Lin, C.-J. (2008). A practical guide to support vector classification. *BJU International*, vol. 101, no. 1, pp. 1396–400. ISSN 1464-410X. 0-387-31073-8.
Available at: http://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf

Hu, X. (2006). PSO Bibliography.
Available at: http://www.swarmintelligence.org/bibliography.php

Hu, X. and Eberhart, R. (2002). Solving constrained nonlinear optimization problems with particle swarm optimization. In: *6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*, pp. 203–206.

Hu, X., Shi, Y. and Eberhart, R.C. (2003). Engineering optimization with particle swarm. *Proceedings of the IEEE Swarm Intelligence Symposium 2003, SIS 2003*, pp. 53–57.

Hu, X., Shi, Y. and Eberhart, R.C. (2004). Recent advances in particle swarm. In: *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 1, pp. 90–97. ISBN 0-7803-8515-2.

Hunter, J.D. (2007 May). Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering*, vol. 9, no. 3, pp. 90–95. ISSN 1521-9615.

Jordehi, A.R. (2015). A review on constraint handling strategies in particle swarm optimisation. *Neural Computing and Applications*, vol. 26, no. 6, pp. 1265–1275. ISSN 0941-0643.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In: *Proceedings of the 1995 IEEE Int. Conf. on Neural Networks held in Perth Australia*, vol. 4, pp. 1942–1948. IEEE Service Center, Piscataway. ISBN 0-7803-2768-3.

Kramer, O. (2010). A review of constraint-handling techniques for evolution strategies. *Applied Computational Intelligence and Soft Computing*, vol. 2010, no. 1, pp. 1–11. ISSN 1687-9724.

Liang, J., Runarsson, T., Mezura-Montes, E., Clerc, M., Suganthan, P. and Coello Coello, C. andDeb, K. (2006). "Criteria for the CEC 2006 special session on constrained real-parameter optimization: Problem definitions and evaluation". Special session on constrained real-parameter optimization, technical report. Tech. Rep., Nanyang Technological University, Singapore.

Lim, D., Ong, Y.S., Setiawan, R. and Idris, M. (2010 August). Classifier-assisted constrained evolutionary optimization for automated geometry selection of orthodontic retraction spring. *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*.

Liu, Z., Li, Z., Zhu, P. and Chen, W. (2018 Oct). A parallel boundary search particle swarm optimization algorithm for constrained optimization problems. *Structural and Multidisciplinary Optimization*, vol. 58, no. 4, pp. 1505–1522. ISSN 1615-1488. Available at: `https://doi.org/10.1007/s00158-018-1978-3`

Marsland, S. (2009). *Machine learning: An algorithmic perspective*. Machine learning & pattern recognition series. Chapman & Hall/CRC Press, Boca Raton. ISBN 978-1-4200-6718-7.

Mezura-Montes, E. and Coello Coello, C.A. (2011). Constraint-handling in nature-inspired numerical optimization: Past, present and future. *Swarm and Evolutionary Computation*, vol. 1, pp. 173–194.

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, pp. 1–32.

Murphy, K.P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press, London, England. ISBN 978-0-262-01802-9.

Neri, F. and Cotta, C. (2012). Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14. ISSN 2210-6502.

Oliphant, T.E. (2006). *A guide to NumPy*. Trelgol Publishing, USA.

Oliphant, T.E. (2007 May). Python for scientific computing. *Computing in Science Engineering*, vol. 9, no. 3, pp. 10–20. ISSN 1521-9615.

Opara, K. and Arabas, J. (2011). Benchmarking procedures for continuous optimization algorithms. *Journal of Telecommunications and Information Technology*, pp. 73–80.

Paquet, U. and Engelbrecht, A.P. (2003). Training support vector machines with particle swarms. *Proceedings of the International Joint Conference on Neural Networks, 2003, IJCNN 2003*, p. 1598.

Parsopoulos, K.E. and Vrahatis, M.N. (2002). Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, vol. 1, no. 2/3, pp. 235–306. ISSN 1567-7818.
Available at: `http://www.springerlink.com/index/4Y89YYY2LEKC4CGC.pdf`

Poli, R. (2008). Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, vol. 2008, no. 3, pp. 1–10. ISSN 1687-6229.
Available at: `http://www.hindawi.com/archive/2008/685175/`

Poli, R., Kennedy, J. and Blackwell, T. (2007). Particle swarm optimization: An overview. *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57. doi: `10.1007/s11721-007-0002-0`.

Regis, R.G. (2014). Particle swarm with radial basis function surrogates for expensive black-box optimization. *Journal of Computational Science*, vol. 5, no. 1, pp. 12–23. ISSN 1877-7503.

Regis, R.G. (2018). Surrogate-assisted particle swarm with local search for expensive constrained optimization. In: Korošec, P., Melab, N. and Talbi, E.-G. (eds.), *Bioinspired Optimization Methods and Their Applications*, pp. 246–257. Springer International Publishing, Cham. ISBN 978-3-319-91641-5.

Roy, R., Hinduja, S. and Teti, R. (2008). Recent advances in engineering design optimisation: Challenges and future trends. *CIRP Annals - Manufacturing Technology*, vol. 57, no. 2, pp. 697–715. ISSN 0007-8506.

Schmitt, B.I. (2015). *Convergence Analysis for Particle Swarm Optimization*. FAU University Press, Erlangen. ISBN 978-3-944057-30-9.

Schoenauer, M. and Michalewicz, Z. (1996). Evolutionary computation at the edge of feasibility. In: *In Proc. of 4th Parallel Problem Solving from Nature*, pp. 245–254. Springer Verlag LNCS.

Schutte, J.F. and Groenwold, A.A. (2003). Sizing design of truss structures using particle swarms. *Structural and Multidisciplinary Optimization*, vol. 25, no. 4, pp. 261–269.

Sedlazek, K. and Eberhard, P. (2006). Using augmented Lagrangian particle swarm optimization for constrained problems in engineering. *Structural and Multidisciplinary Optimization*, vol. 32, pp. 277–286.

Shi, Y. and Eberhart, R.C. (1998). A modified particle swarm optimizer. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 1998*, pp. 69–73.

Song, M. and Gu, G. (2004 August). Research on particle swarm optimization: a review. *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, vol. 4, pp. 2236–2241.

Sörensen, K. (2015). Metaheuristics - the metaphor exposed. *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18. ISSN 0969-6016.
Available at: `https://onlinelibrary.wiley.com/doi/epdf/10.1111/itor.12001`

Sörensen, K. and Glover, F. (2013). Metaheuristics. In: Gass, S. and Fu, M. (eds.), *Encyclopedia of Operations Research and Management Science*, 3rd edn, pp. 960–970. Springer US.

Sörensen, K., Sevaux, M. and Glover, F. (2018). A history of metaheuristics. In: Martí, R., Pardalos, P.M. and Resende, M.G.C. (eds.), *Handbook of Heuristics*, pp. 791–808. Springer International Publishing, Cham. ISBN 978-3-319-07124-4.

Steinwart, I., Hush, D. and Scovel, C. (2006). An explicit description of the reproducing Kernel Hilbert spaces of Gaussian RBF kernels. *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4635–4643. ISSN 0018-9448.

Sun, C. and Jin, Y. (2015). A two-layer surrogate-assisted particle swarm optimization algorithm. *Soft Computing*, vol. 19, no. 6, pp. 1461–1475.

Sun, C., Jin, Y., Cheng, R., Ding, J. and Zeng, J. (2017). Surrogate-assisted cooperative swarm optimization of high-dimensional expensive problems. *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 4, pp. 644–660. ISSN 1089-778X.

Surjanovic, S. and Bingham, D. (2013). Virtual library of simulation experiments: Test functions and datasets. Retrieved November 10, 2018, from `http://www.sfu.ca/ ssurjano`.

Tiwari, A., Hoyos, P.N., Hutabarat, W., Turner, C., Ince, N., Gan, X.-P. and Prajapat, N. (2015). Survey on the use of computational optimisation in UK engineering companies. *CIRP Journal of Manufacturing Science and Technology*, vol. 9, pp. 57–68. ISSN 1755-5817.

Vanderplaats, G.N. (2011). *DOT User's Manual*. Vanderplaats Research & Development, Inc., 1767 S. 8th Street, Suite 200 Colorado Springs, CO 80905.
Available at: `http://www.vrand.com`

Vempati, S., Vedaldi, A., Zisserman, A. and Jawahar, C.V. (2010). Generalized RBF feature maps for Efficient Detection. *Proceedings of the British Machine Vision Conference 2010*.
Available at: `http://www.bmva.org/bmvc/2010/conference/paper2/index.html`

Venter, G. (2010). Review of optimization techniques. In: *Encyclopedia of Aerospace Engineering*, pp. 4728–4736. John Wiley & Sons, Ltd.

Venter, G. and Haftka, R.T. (2010). Constrained particle swarm optimization using a bi-objective formulation. *Structural and Multidisciplinary Optimization*, vol. 40, pp. 65–76.

Venter, G. and Sobieszczanski-Sobieski, J. (2004). Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization. *Structural and Multidisciplinary Optimization*, vol. 26, no. 1-2, pp. 121–131. ISSN 1615-147X.

Wang, G. and Shan, S. (2007). Review of metamodeling techniques in support of engineering design optimization. *Journal of Mechanical design*, vol. 129, no. 4, pp. 370–380.

Wang, Y., Yin, D.Q., Yang, S. and Sun, G. (2018). Global and local surrogate-assisted differential evolution for expensive constrained optimization problems with inequality constraints. *IEEE Transactions on Cybernetics*, pp. 1–15. ISSN 2168-2267.

While, L. and Hingston, P. (2013). Usefulness of infeasible solutions in evolutionary search: An empirical and mathematical study. *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, pp. 1363–1370.

Wilke, D.N., Kok, S. and Groenwold, A.A. (2007*a*). Comparison of linear and classical velocity update rules in particle swarm optimization: Notes on diversity. *International Journal for Numerical Methods in Engineering*, vol. 70, pp. 962–984.

Wilke, D.N., Kok, S. and Groenwold, A.A. (2007*b*). Comparison of linear and classical velocity update rules in particle swarm optimization: Notes on scale and frame invariance. *International Journal for Numerical Methods in Engineering*, vol. 70, pp. 985–1008.

Wise, J.N. (2008). *Optimization of a Low Speed Wind Turbine using Support Vector Regression*. Master's thesis, Department of Mechanical Engineering, University of Stellenbosch.

Witt, C. (2011). Theory of particle swarm optimization. In: Auger, A. and Doerr, B. (eds.), *Theory of Randomized Search Heuristics*, chap. 7, pp. 197–223. World Scientific Publishing.

Wolpert, D.H. and Macready, W.G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82. ISSN 1089778X.

Yang, Z., Qiu, H., Gao, L., Cai, X., Jiang, C. and Chen, L. (2018). A surrogate-assisted particle swarm optimization algorithm based on efficient global optimization for expensive black-box problems. *Engineering Optimization*, pp. 1–18. ISSN 1029-0273.
Available at: `https://doi.org/0305215X.2018.1477940`

Zhang, Y., Wang, S. and Ji, G. (2015). A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical Problems in Engineering*, pp. 1–38.

Appendices

Appendix A

Python2 implementation additional information

The algorithm flow diagrams for concepts 1 to 4 are given in figures A.1, A.2, A.3 and A.4.

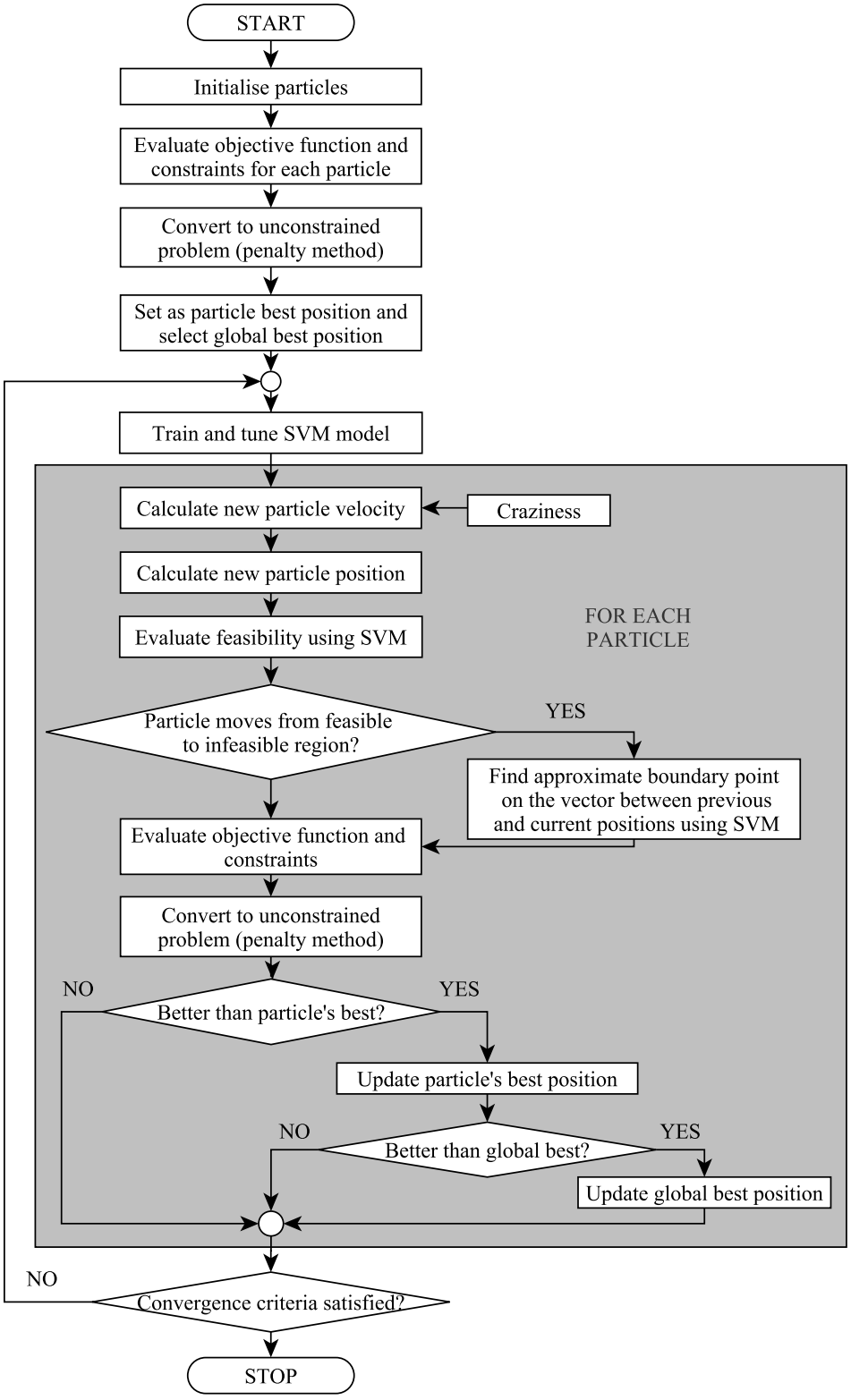

Figure A.1: Concept 1 flow diagram

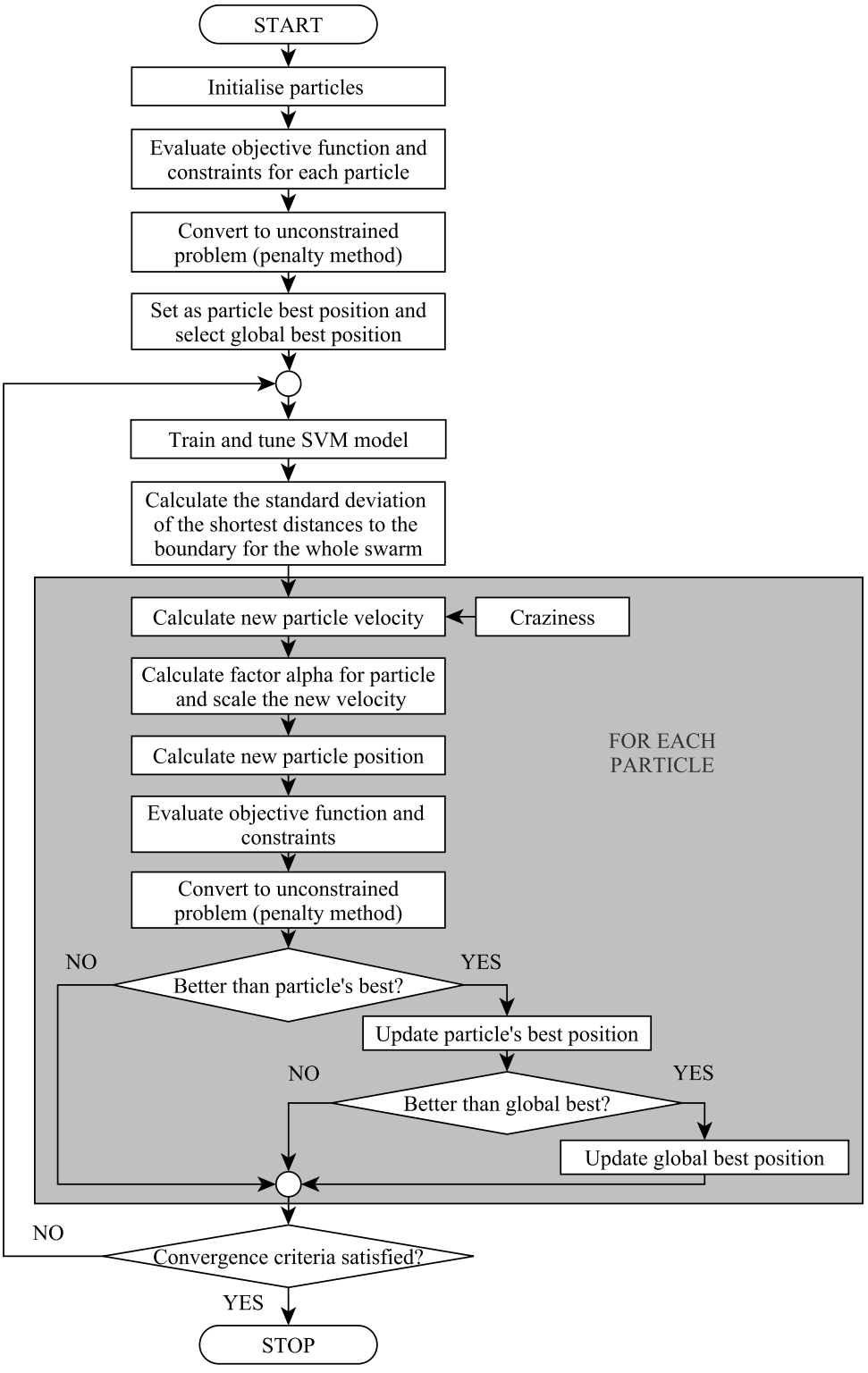

Figure A.2: Concept 2 flow diagram

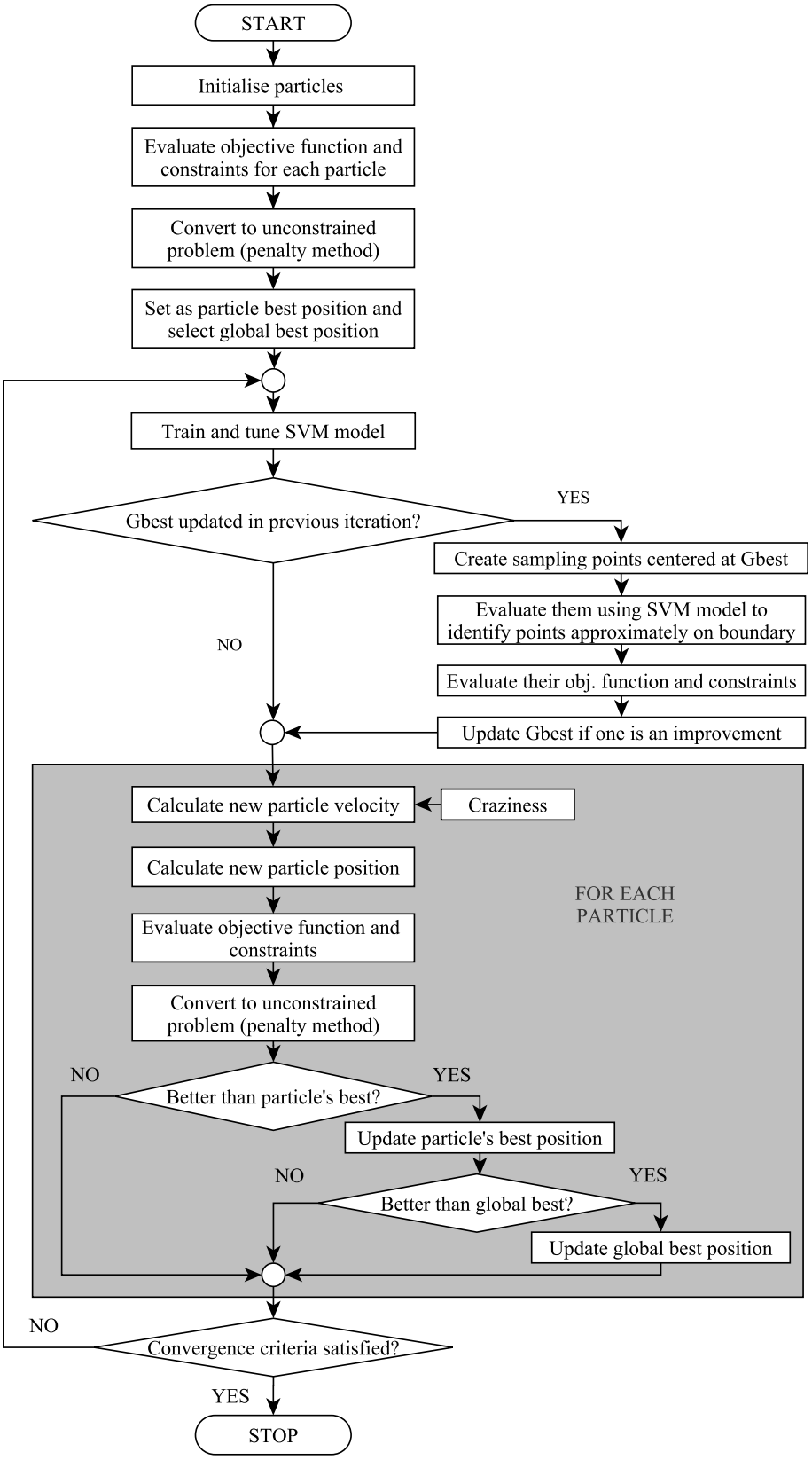

Figure A.3: Concept 3 flow diagram

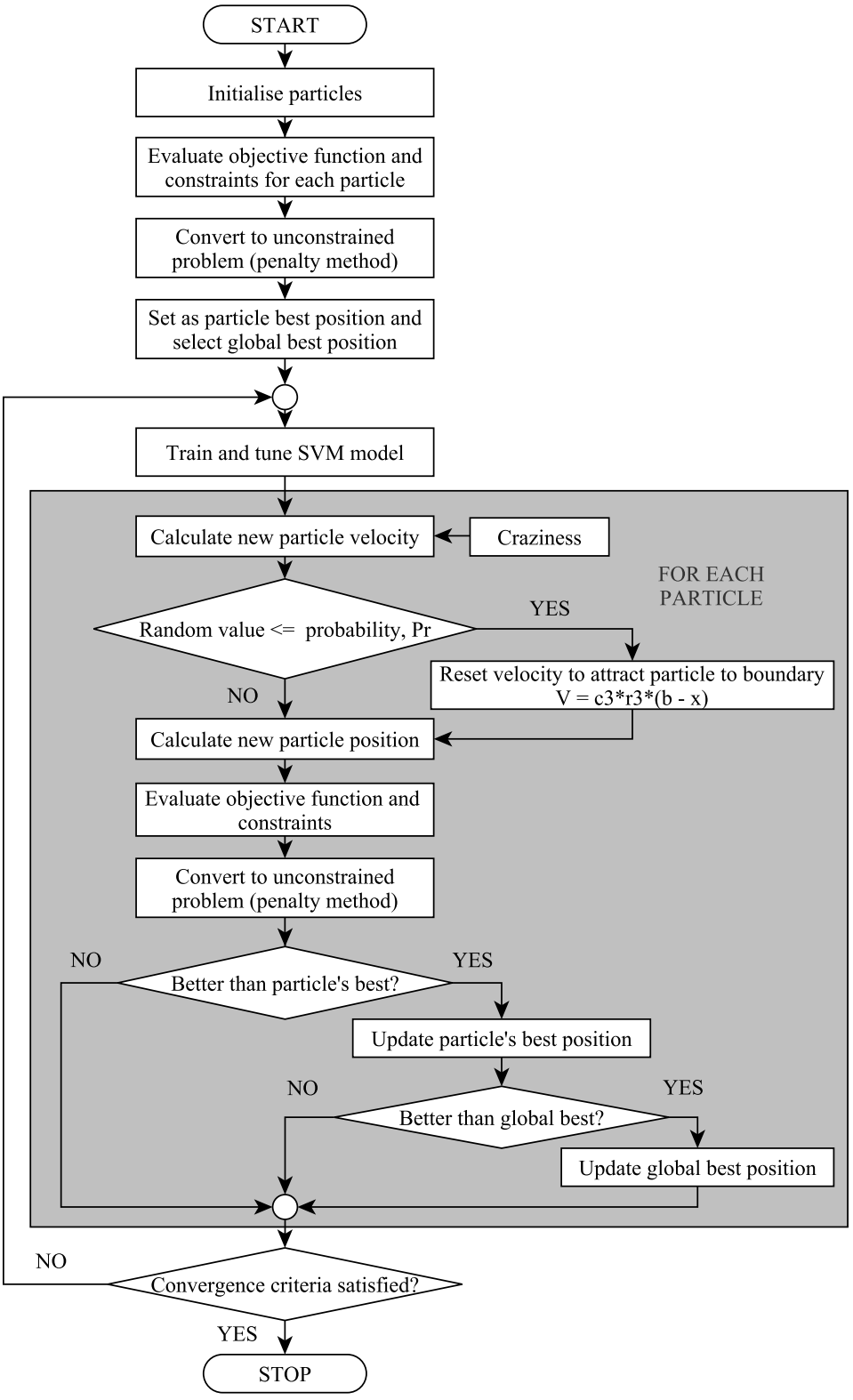

Figure A.4: Concept 4 flow diagram