Metaheuristics for Black-Box Robust Optimisation Problems

Martin Hughes

A thesis presented for the degree of

Doctor of Philosophy

Department of Management Science

Lancaster University

United Kingdom

May 2020

Abstract

Our interest is in the development of algorithms capable of tackling robust black-box optimisation problems, where the number of model runs is limited. When a desired solution cannot be implemented exactly (implementation uncertainty) the aim is to find a robust one. Here that is to find a point in the decision variable space such that the worst solution from within an uncertainty region around that point still performs well.

This thesis comprises three research papers. One has been published, one accepted for publication, and one submitted for publication. We initially develop a single-solution based approach, largest empty hypersphere (LEH), which identifies poor performing points in the decision variable space and repeatedly moves to the centre of the region devoid of all such points. Building on this we develop population based approaches using a particle swarm optimisation (PSO) framework. This combines elements of the LEH approach, a local descent directions (d.d.) approach for robust problems, and a series of novel features. Finally we employ an automatic generation of algorithms technique, genetic programming (GP), to evolve a population of PSO based heuristics for robust problems. We generate algorithmic sub-components, the design rules by which they are combined to form complete heuristics, and an evolutionary GP framework. The best performing heuristics are identified.

With the development of each heuristic we perform experimental testing against comparator approaches on a suite of robust test problems of dimension between 2D and 100D. Performance is shown to improve with each new heuristic. Furthermore the generation of large numbers of heuristics in the GP process enables an assessment of the best performing sub-components. This can be used to indicate the desirable features of an effective heuristic for tackling the problem under consideration. Good performance is observed for the following characteristics: inner maximisation by random sampling, a small number of inner points, particle level stopping conditions, a small swarm size, a Global topology, and particle movement using a baseline inertia formulation augmented by LEH and d.d. capabilities.

Declaration

This thesis is my original work and has not been submitted, in whole or in part, for a degree at this or any other university. To the best of my knowledge this thesis does not contain any material published or written by another person, other than as acknowledged in the text. A statement of authorship for the three papers is given in Section 1.4.2.

Acknowledgements

I would like to thank my three supervisors, Prof. Marc Goerigk, Dr. Vikram Dokka, and Prof. Michael Wright.

I am extremely grateful to Marc for the level of support he has given me. I could not have asked for better supervision, which was at exactly the right level to allow me to pursue areas of interest whilst keeping on course – both technically and in terms of completing my research.

I would also thank Vikram and Mike for their supervision and support, as well as for their critical input and positive guidance.

Thank you to everyone in the Lancaster University Department of Management Science whose involvement, encouragement and input has made this possible. Finally, thank you to the Engineering and Physical Sciences Research Council who funded this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

A model is an imitation of a real life problem, an approximation of reality. Models are extensively used to improve understanding, aid investigation or support informed decision making in a wide range of situations. One important application of such models is the identification of optimal solutions, that is the identification of the model input variable values that produce the best outputs. However the size and nature of the decision variable solution space, and the model run time, may make a comprehensive – exhaustive or simply extensive – evaluation of the problem space, and so a direct identification of optima, computationally infeasible. In such cases an efficient approach is required to search for global optima.

Mathematical programs such as linear or mixed-integer programs, are one form of model that are explicitly formulated as optimisation problems: the model representation imposes assumptions on the structure of the decision variable space and objective function. Such models are well suited to efficient solution, and identification of global optima may be theoretically guaranteed when feasible solutions exist. However many real-world problems are not suited to expression as a mathematical program, for example a solution may be evaluated by a simulation tool. It is this form of model, where no assumptions are made about the model structure, that is of concern here.

From an optimisation perspective such a model can be thought of as a black-box, where decision variable values are input and outputs generated for interpretation as an objective, see Figure 1.1. For this reason optimisation search techniques that can be applied to such general problems, or indeed to large computationally infeasible mathematical programs, are of interest here. Specifically our concern is with metaheuristics, a class of general, rule-based search techniques.

An additional widespread feature of many real-world problems is the consideration of uncertainty which may impact on model outputs, and so on corresponding objective function values. One strategy is to simply ignore any uncertainty and perform a standard search, possibly assessing and reporting on the sensitivity of the optimum after it

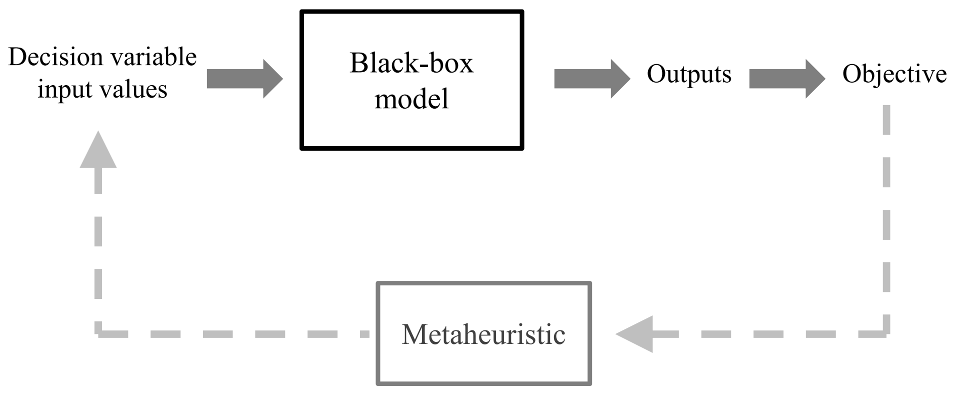


Figure 1.1: With black-box models no assumptions are made about the model structure. For optimisation, decision variable values are input and outputs generated for interpretation as an objective.

has been identified. However it has been established that optimal solutions which are sensitive to parameter variations within known bounds of uncertainty may substantially degrade the optimum objective function value, meaning that solutions sought without explicitly taking account of uncertainty are susceptible to significant sub-optimality, see [BTEGN09, GS16]. In the face of uncertainty the focus of attention for an optimisation analysis shifts from the identification of a solution that just performs well in the expected case, to a solution that performs well over a range of scenarios.

When considering uncertainty in decision making problems a frequent distinction is made between model, or parameter, uncertainty (where the problem data is not known exactly) and implementation uncertainty (where a decision cannot be put into practice with full accuracy). Implementation uncertainty is also known as decision uncertainty, and implementation errors can be thought of as perturbations or changes that could effect a solution after it has been identified. Therefore it can be recognized that an optimal solution that is somewhat insensitive to such perturbations is desirable, see [BTEGN09, Tal09, BNT10b]. Only implementation uncertainty is considered explicitly here.

Specifically we develop new algorithms for black-box global optimisation problems taking account of implementation uncertainty. A situation where an ideal solution cannot be achieved exactly is common in many real-world applications. For example in engineering, manufacturing or construction it may not be possible to meet design specifications exactly, in which case a solution is sought which is tolerant of some variation in the design variables. In scheduling, timetabling, logistics, queuing or supply chain management it is desirable to be able to tolerate some deviation in resource levels and transportation or processing times. See, for example, [PBJ06, BNT07, BNT10b, Kru12, GMT14, GS16].

In Section 1.2 we formalise the problem of concern here. Section 1.3 is a literature review with specific focus on metaheuristics for robust optimisation. This includes an explanation of two techniques that play recurring roles in the heuristics developed here, the local robust descent directions approach [BNT07, BNT10b, BNT10a] and the 'standard' (non-robust) particle swarm optimisation heuristic [KE95, KES01, Tal09], Sections 1.3.4 and 1.3.5.

The focus of the work described in this thesis is the development of new metaheuristics for robust problems, which are tested and shown to exhibit improved performance. In Section 1.4 we provide overviews of the three papers setting out this work, including an explanation of the author contributions. The three papers are presented in Chapters 2 to 4. Whilst all three papers focus on developing new improved metaheuristics, the third uses genetic programming to evolve populations of heuristics for robust problems and so enables an additional assessment of those algorithmic sub-elements of a heuristic included in the most effective search approaches. We end with a summary and conclusions in Chapter 5.

It should be noted that whilst within each chapter the notation used is consistent, due to the three-paper format of this thesis the notation across chapters may vary. For example, whilst the min max problem which is the focus of our work uses the reference (MinMax) both here in Chapter 1 and in Chapter 4, in Chapter 2 the reference is (ROP) and in Chapter 3 it is (MM).

1.2 Problem description

We consider a general optimisation problem of the form:

$$\begin{aligned} &\min f(\boldsymbol{x}) \\ &\text{s.t. } \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

Here $\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T$ denotes the n-dimensional vector of decision variables, $f : \mathbb{R}^n \to \mathbb{R}$ is the single objective function, and $\mathcal{X} \subseteq \mathbb{R}^n$ is the set of feasible solutions. Writing $[n] := \{1, \dots, n\}$, we assume that problems are box-constrained, i.e. the decision variable space is contained within lower and upper bounds: $\mathcal{X} = \prod_{i\in[n]}[l_i, u_i]$. We further assume that any other potential feasibility constraints are ensured through a penalty in the objective.

Such a problem, without any consideration of uncertainty, is designated the nominal problem here. Consider for example the non-convex one dimensional problem due to

[Kru12]:

$$
\begin{aligned}
f(x) &= 1 + f_1(x) + f_2(x),\\
f_1(x) &= \begin{cases} (\frac{x-4}{5})^2 & \text{if } x < 4,\\ 1 & \text{otherwise}\end{cases}\\
f_2(x) &= -1.8\exp\left(-\frac{(x-5)^2}{0.2}\right) - 2\exp\left(-\frac{(x-7)^2}{0.1}\right)
\end{aligned}
$$

For the nominal problem shown in Figure 1.2, some standard metaheuristic could be used to search for the global minimum at $\boldsymbol{x}_0$ – recognising that metaheuristic approaches may not be able to precisely locate that global minimum. However if the solution that a decision maker wants to implement, $\boldsymbol{x}$, may be somewhat perturbed in practice, the potential impact on the identification of the global minimum needs to be taken into consideration. The sensitivity of the objective to variations in $\boldsymbol{x}$ in the region of $\boldsymbol{x}_0$ is of particular concern, as highlighted in Figure 1.3.

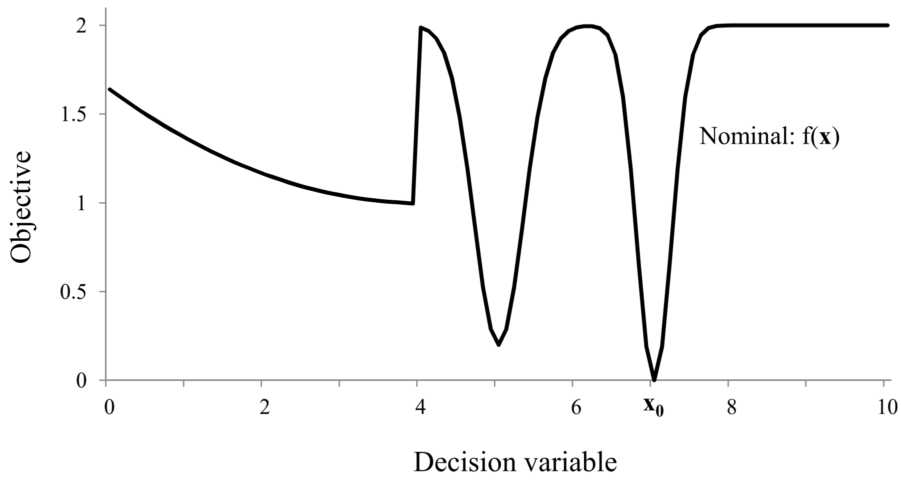


Figure 1.2: One dimensional problem due to [Kru12]. The global optimum for the nominal problem is at $\boldsymbol{x}_0$.

Uncertainty can be included in the problem formulation as:

$$
\begin{aligned}
&\min\ f(\boldsymbol{x}, \boldsymbol{\xi})\\
&\text{s.t. } \boldsymbol{x} \in \mathcal{X}(\boldsymbol{\xi})
\end{aligned}
$$

where $\boldsymbol{\xi}$ represents the uncertainty in the problem.

If this problem is to be tackled using stochastic optimisation techniques, the probability distributions over all possible scenarios $\boldsymbol{\xi}$ must be known. However if it is only assumed that some set $\mathcal{U}$ is identified containing all possible uncertainty scenarios (potentially infinite in number), the problem is one of robust optimisation. Here only the

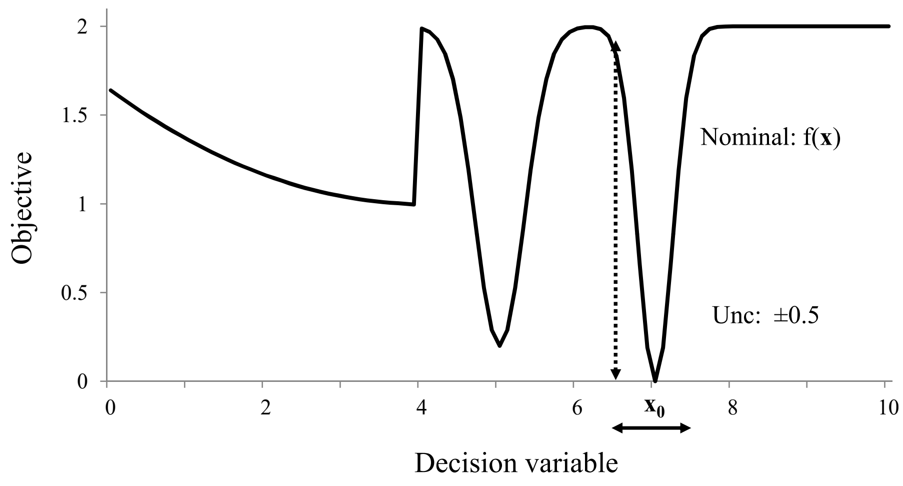


Figure 1.3: With the consideration of uncertainty only a 'close' solution may be realised, which is of particular concern in the region of $\boldsymbol{x}_0$; problem due to [Kru12].

robust setting is of interest, and we consider the case where instead of being able to put a desired solution $\boldsymbol{x}$ into practice with full accuracy, only a 'close' solution $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \Delta\boldsymbol{x}$ may be realised.

Specifically the focus here is the classic robust approach, that is to find a robust solution $\boldsymbol{x}$ such that for any such $\tilde{\boldsymbol{x}}$ from the uncertainty neighbourhood of $\boldsymbol{x}$, the worst case performance is optimised. Here we further follow the setting of [BNT10b] and assume that $\Delta\boldsymbol{x}$ lies in the so-called uncertainty set:

$$\mathcal{U} := \{\Delta\boldsymbol{x} \in \mathbb{R}^n \mid \|\Delta\boldsymbol{x}\| \leq \Gamma\}$$

where $\Gamma > 0$ defines the magnitude of the uncertainty, specifically the radius of the uncertainty neighbourhood around a point. $\|\cdot\|$ refers to the Euclidean norm.

Using a local maximisation to find a robust solution $\boldsymbol{x}$, the worst case value $g(\boldsymbol{x})$ is optimised for any $\tilde{\boldsymbol{x}}$ in the uncertainty neighbourhood of $\boldsymbol{x}$:

$$g(\boldsymbol{x}) := \max_{\Delta\boldsymbol{x} \in \mathcal{U}} f(\boldsymbol{x} + \Delta\boldsymbol{x})$$

Making the complete robust problem one of finding the outer minimum objective in $\mathcal{X}$, where that objective is itself an inner maximisation in the uncertainty neighbourhood around each solution $\boldsymbol{x} \in \mathcal{X}$ for the nominal objective function:

$$\min_{\boldsymbol{x} \in \mathcal{X}} g(\boldsymbol{x}) = \min_{\boldsymbol{x} \in \mathcal{X}} \max_{\Delta\boldsymbol{x} \in \mathcal{U}} f(\boldsymbol{x} + \Delta\boldsymbol{x}) \tag{MinMax}$$

This type of problem is known as min max and it is the solution of this problem (MinMax) that is the focus of our work.

Applying this to the 1D non-convex problem [Kru12], Figure 1.4 shows the worst case cost or min max value (dashed grey) at any individual point $\boldsymbol{x}$ as determined by assessing the uncertainty neighbourhood around that point, in order to identify the maximum value within that uncertainty neighbourhood. Then within the global minimisation search the nominal objective is superseded by the worst case cost. That is we seek the global minimum for the worst case cost due to implementation uncertainty of ± 0.5 on the corresponding nominal curve.

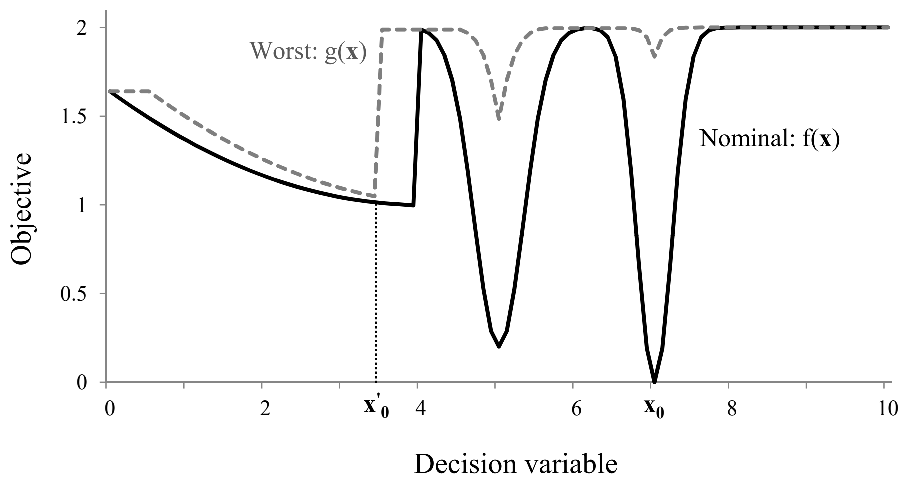


Figure 1.4: The worst case cost curve (dashed grey) is generated by determining the maximum objective value in the uncertainty neighbourhood around all points $\boldsymbol{x}$ on the nominal (solid black) curve. Due to the uncertainty the global optimum shifts to $\boldsymbol{x}_0'$; problem due to [Kru12].

In Figure 1.4 it can be observed that the global optimum has shifted to $\boldsymbol{x}_0'$, and that there is a reduction in the optimum objective. The difference between the nominal and robust optimal objective function values is the 'price of robustness', see [BS04]. It should also be noted that if we were to ignore the implementation uncertainty and simply accept $\boldsymbol{x}_0$ as the global optimum, we risk the possibility of a very poor outcome: the worst case cost (dashed grey) objective function value at $\boldsymbol{x}_0$. Whilst $f(\boldsymbol{x}_0') > f(\boldsymbol{x}_0)$, $g(\boldsymbol{x}_0') < g(\boldsymbol{x}_0)$.

Note that $\boldsymbol{x} + \Delta\boldsymbol{x}$ may not be in $\mathcal{X}$, for which reason we assume that the definition of f is not limited to $\mathcal{X}$. However, if it is desired that $\boldsymbol{x} + \Delta\boldsymbol{x} \in \mathcal{X}$ for all $\Delta\boldsymbol{x} \in \mathcal{U}$ (strictly robust), then this can be ensured by reducing the size of the feasible search space by Γ.

One further restriction will be assumed to apply here in a general sense. It is assumed that due to practical issues there will be some limitation on the number of function evaluations (model runs) that can be employed in a robust search. Within the experimental testing in the research described in Chapters 2 to 4, budget limitations are assumed on the number of test function evaluations that can be undertaken in a single global robust search. Such budgetary restrictions lead to some implicit trade-off between the extent of each inner search and the performance of the outer global search, see [MLM15, BL15, LBR16, EDHX17].

1.3 Literature review

1.3.1 Metaheuristics

One way to tackle hard optimisation problems is to start with a baseline solution and try to improve on it. Metaheuristics involve the application of problem independent heuristics to iteratively improve one or many solution candidates. A good metaheuristic both explores the search space in the neighbourhood of already-identified good solutions and stochastically examines unvisited regions. Such an approach forestalls misconvergence to local rather than global optima, see [Tal09, Luk13, BLS13]. Although metaheuristics are not guaranteed to find globally optimal solutions they have been widely shown to find good solutions in a reasonable amount of time, even in large search spaces [EHG05, Luk13].

The field of metaheuristics is extensive. One high-level categorisation is between single-solution based and population based approaches. Single-solution approaches move from a single initial solution, following some to-be-determined trajectory through the search space. At each step the objective function evaluation is undertaken, and based on the information gained it is determined where in the decision variable space to move to next. A number of such approaches and variations exist, including hill-climbing and gradient descent techniques, local search techniques, tabu search, and simulated annealing; see [Tal09, Luk13, BLS13].

The search for global optima is enhanced by population based algorithms, which explore the solution space in multiple locations at a time. Two high-level categories of population approaches are evolutionary algorithms and swarm intelligence based algorithms. Evolutionary algorithms simulate natural evolution, with a population of individuals evolving over time. Each individual represents a solution, a point in the decision variable space. The objective function for each individual is evaluated and then interpreted as a measure of the relative fitness of the individual. A new population (generation) is constructed through fitness-based selection of individuals from the current population and some combination and modification of those individuals. This development from one generation to another repeats. In this way individuals are evolved that are well-adapted to their environments, that is they are good (fit, optimal) solutions. Amongst the most commonly cited evolutionary optimisation algorithms are genetic algorithms (GAs) and differential evolution, see [Mit98, SP97, EHG05, Tal09].

Swarm intelligence algorithms simulate interactions between individuals and with their environment, with gained information contributing to the collective intelligence of the group. Again each individual represents a position in the decision variable space. The evaluated objective function values provide information on the effectiveness of individual solutions, which feeds into a pool of information to be accessed by some or all of the individuals. The network topologies through which such information is shared forms an important component of swarm based optimisation heuristics. Based on a combination of their own and the collective information, individuals move to new locations in the decision variable space. This process is applied to each individual and repeats for the entire population over a number of iterations. Established algorithms include particle swarm optimisation (PSO) and ant colony optimisation, see [KE95, KES01, RC13]. These techniques mimic complex, self-organising systems of individuals. What emerges from their interactions corresponds to an efficient exploration of a solution space [KES01].

Many methods and hybrids-variants have been developed, see [KE95, SP97, Mit98, EHG05, RC13]. Each algorithm has its strengths and weaknesses. Often the use of a specific technique may be somewhat problem and context dependent. There is no single algorithm that can solve all problems better than all other algorithms [Luk13, PHP12].

1.3.2 Optimisation under uncertainty

Stochastic optimisation is a commonly used approach to the incorporation of uncertainty for black-box problems, when there is a knowledge of the probability distributions of the uncertain parameters. There some statistical measure of the fitness of a solution is assessed in the neighbourhood of a point in the decision variable space, e.g. using Monte Carlo simulation. The optimisation search is based on the use of that measure as the objective value. The estimated statistical measure may be the expected value, or a more elaborate model such as the variance in the fitness of a solution, or may even take the form of more than one measure in a multi-objective optimisation setting, see [PBJ06, HdMB14].

A substantial amount of work has been undertaken on the development of meta-heuristics for stochastic optimisation, including, for example on the recognition of and exploitation of the inherent stochasticity of evolutionary algorithms, [TG97, Bra98, BSS01, PBJ06, BS07, Kru12]. A common feature of such approaches is that they are general, and can be applied to any situation where knowledge of the uncertain parameters' probability distributions is assumed and objective functions can be evaluated.

Here, however, the focus is robust optimisation. As described in Section 1.2, in this case it is only assumed that some set can be identified which contains all possible uncertainty scenarios. Specifically our focus is the classic robust approach of finding a solution across all scenarios that is always feasible (strictly robust) and optimises its performance in the worst case: that is min max as defined by (MinMax).

Robust optimisation is a relatively young field, whose modern form was first developed in [KY97] and [BTN98]. Overviews of the field can be found in [BTEGN09, ABV09, GS16]. The definition of the uncertainty set $\mathcal{U}$ and the measure of performance (optimal in the worst case for min max) are the critical components of a robust optimisation analysis. Two basic ways of describing $\mathcal{U}$ are explicitly, in the discrete case, and in terms of a parameter interval such that any value between a lower and an upper bound is feasible, [ABV09]. [BBC11] consider several classes of uncertainty, raising the notion of a 'budget of uncertainty' for trading-off robustness and performance.

The field of robust optimisation has been primarily aligned with mathematical programming approaches. There the methodology is based around the definition of reasonable uncertainty sets and the reformulation of computationally tractable mathematical programming problems. For specific forms of convex optimisation problems the problem incorporating uncertainty can be re-formulated to another tractable, convex problem, see [BNT07, GS10].

To overcome concerns that the strictly robust worst case approach may be overly conservative, the concept of robustness can be expanded in terms of both the uncertainty set considered and the robustness measure [GS16]. On the assumption that it is overly pessimistic to assume that all implementation errors take their worst value simultaneously [BS04] consider an approach where the uncertainty set is reduced, and a robust model defined where the optimal solution is required to remain feasible for uncertainty applied to only a subset of the decision variables at any given time. Min max regret, see [ABV09], is an alternative to min max, seeking to minimise the maximum deviation between the value of the solution and the optimal value of a scenario, over all scenarios. [BTBB10] considers soft robustness, which utilises a nested family of uncertainty sets. The distributionally robust optimisation approach, see [GS10], attempts to bridge robust and stochastic techniques by utilising uncertainty defined as a family of probability distributions, seeking optimal solutions for the worst case probability distribution. [CG16] use a bi-objective approach to balance average and worst case performance by simultaneously optimising both.

A number of authors have also developed approaches based around the recognition that decisions in the face of uncertainty, rather than being static one-off events resulting in fully determined outcomes, may actually provide some opportunity for recourse. [BTGGN03] and [CZ09] developed approaches based on the recognition that for multiple-stage problems a subset of the decisions could be made after some or all of the uncertainties had been realised, see [GS10]. [BTGGN03] introduced adjustable robustness which divides the variables into separate sets of those that have to be evaluated immediately in the face of uncertainty and those that can be evaluated once the uncertainty has been resolved. A similar two-stage approach, recovery robustness is considered for example in [Goe12].

Robust optimisation in a mathematical programming context has been application-driven, so considerable work has been undertaken in applying robustness techniques to specific problems or formulations, see [BS07, GS16]. But it is robust optimisation as applied to general black-box problems that is of interest here, where the model is not simply defined through algebraic functions. There has been some cross-over into the application of specific heuristics, for example see [GLT97, AVCMSdCM11]. However application to general problems has been less well addressed [GS16]. Robust approaches applied to black-box models are much less widely considered than approaches for mathematical programming problems, see [MWPL13, GS16, MWPL16]. Relatively recently, robust optimisation with implementation uncertainty has also been extended to multi-objective optimisation, see [EKS17].

1.3.3 Metaheuristics for robust optimisation

Given a situation where a mathematical programming approach does not apply, the solution of (MinMax) can be tackled with standard metaheuristic techniques applied to both the inner maximisation and outer minimisation problems. In co-evolutionary approaches two populations (or swarms) evolve separately but are linked, so the fitness of individuals in one group is informed by the individuals in the other, see [CSZ09]. Techniques include two-population genetic algorithm approaches [Her99, Jen04], and two-swarm co-evolutionary particle swarm optimisation approaches [SK02, MKA11]. However while a brute force co-evolutionary approach is technically acceptable, in practice using complete inner maximisation searches to generate robust values for each individual in each generation of the outer minimisation is likely to be expensive in terms of model runs (function evaluations), see [MWPL16]. Instead more practical co-evolutionary approaches require the application of additional simplifications and assumptions, for example using only small numbers of populations for the outer search and the inner (uncertainty) search which share information between populations from generation to generation, or following several generations, see [CSZ09, MKA11].

One general area of research is the use of emulation (surrogates or meta-models) alongside true objective function evaluations to reduce the potential burden of computational run times and the number of model-function evaluations, see [BS07, KVDHL16]. [ZZ10] use a surrogate-assisted evolutionary algorithm to tackle the inner search for black-box min max problems. [MWPL13, MWPL16] employs Kriging meta-modelling coupled with an expected improvement (EI) metric, as well as a relaxation of the inner maximisation search. The EI metric is used to efficiently choose points in the decision variable space where nominal (expensive) function evaluation should be undertaken, here with a view to most efficiently improving the estimate of the robust global minimum. This is known as Efficient Global Optimisation (EGO), see [JSW98]. The relaxation involves iteratively performing the outer minimisation on a limited inner uncertainty neighbourhood followed by an extensive inner maximisation search in the region of the identified outer minimum. This continues whilst the inner search sufficiently deteriorates the outer solution, with the inner maximum point being added to the limited inner uncertainty set with each iteration.

A second approach due to [uRLvK14, uRL17] also uses Kriging and an EI metric, building on a meta-model of the expensive nominal problem by applying a robust analysis directly to the Kriging model and exploiting the fact that many more inexpensive function evaluations can be performed on this meta-model. This therefore requires the determination of the robust global minimum of the Kriging meta-model. A modified EI metric is then calculated for the worst case cost function of the meta-model, to efficiently guide the search in the nominal expensive function space to the identification of the global maximum value of the modified EI metric – which is itself a function of the worst case cost function of the meta-model. The algorithm iterates, given a new nominal function evaluation the meta-model is recalculated, its global robust minimum determined, and the maximum EI value identified, up to some stopping criteria. On the basis that a highly accurate meta-model is a requirement, and that both the calculation of the robust global minimum for the meta-model and the associated identification of the EI maximum feed in to the evolution of the meta-model, it is reasonable to assume that both of these stages are themselves required to produce results to some degree of accuracy.

One alternative emulation approach is to employ Bayesian techniques. For example [CLSS17] is an approach designed to be applicable to general problems and actually uses a mathematical programming approach. However the mathematical program is not applied directly to general problems but rather to an emulated version of the problem, an approximate Bayesian oracle. Alternatively [SEFR19] employ a Bayesian approach for very expensive-to-evaluate functions, applying it to test problems of up to 10 dimensions using only small numbers of function evaluations.

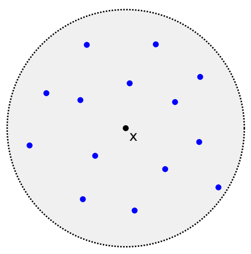
However the primary challenge with current emulation based approaches is their application to problems other than those of relatively low dimension. This is due to an inability to generate accurate emulation models, and so the emulation based papers discussed here have either been restricted to low dimensional non-convex problems, or simpler convex and convex-concave problems. Or in the case of [CLSS17] it is simply assumed that a Bayesian oracle is available.

We end the literature section with some more detailed descriptions of two existing techniques which have informed our work, and which are used as both comparator heuristics in the experimental testing and as a basis for elements of our novel heuristics. The first is descent directions (d.d.) [BNT07, BNT10b, BNT10a], a local technique for robust problems which addresses (MinMax) and can be applied to black-box problems with no explicit additional restrictions. The second heuristic is particle swarm optimisation (PSO) [KE95, KES01, Tal09]. This is not a robust approach, although it can be extended to one through the inclusion of an inner maximisation layer.

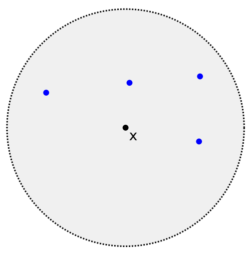
1.3.4 Descent directions

Descent directions [BNT07, BNT10b, BNT10a] is an individual based approach whereby a search progresses by iteratively moving along 'descent directions'. Uncertainty around individual points is assessed (e.g. by using local gradient ascents in [BNT10b]), based on which undesirable ‘high cost points’ (hcps) are identified. Steps are taken in directions which point away from these hcps, until no direction can be found. At that stage it is assumed that a robust local minimum has been reached. The approach can be simplistically extended to a proxy global search by using random re-starts: once the heuristic stops at a robust local minimum, a new local search is started from a randomly selected start point. This may repeat until some budget of function evaluations has been exhausted.

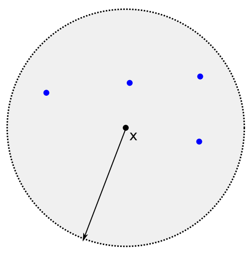
The description of the d.d. heuristic given below is taken from [BNT10b]. In the research described in Chapters 2 to 4 a randomly re-starting version of this heuristic is used as a comparator, in conjunction with stated budgets of function evaluations. The form and extent of the inner maximisation search and other parameter settings, typically determined by tuning for the comparator d.d. employed in Chapters 2 to 4, are described there. In the description of the heuristic here it can be assumed that some inner maximisation heuristic is used to approximate uncertainty neighbourhood worst case costs. Parameter values stated below are as in [BNT10b], and are given as indicative.



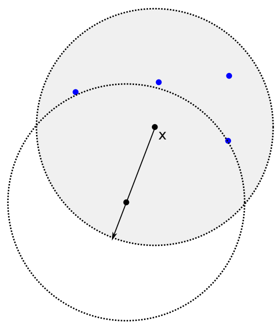
(a) Candidate point $\hat{\boldsymbol{x}}$ (centre), and points evaluated for the inner maximisation problem (blue).



(b) Subset $H(\hat{\boldsymbol{x}})$ of critical high cost points.



(c) A descent direction is identified by solving a second order cone problem.



(d) The step size is determined.

Figure 1.5: Description of the descent direction robust local search approach [BNT10b].

The heuristic is initialised by randomly sampling a first candidate point $\boldsymbol{x}(0)$ at step $t = 0$. Around any given candidate $\boldsymbol{x}(t)$ at step t, and starting with $\boldsymbol{x}(0)$, an inner maximisation is performed. See Figure 1.5a. From then on all function evaluations up to and including the current step (candidate point) t are recorded in a history set $H(t)$. The combination of the function evaluations undertaken in the inner maximisation around any given candidate $\boldsymbol{x}(t)$, plus any additional points in the Γ-uncertainty neighbourhood of $\boldsymbol{x}(t)$ previously recorded in H, provide a knowledge of the uncertainty neighbourhood $N(\boldsymbol{x}(t)) = \{\boldsymbol{x}(t) + \Delta\boldsymbol{x}(t) \mid \Delta\boldsymbol{x}(t) \in \mathcal{U}\}$ around candidate $\boldsymbol{x}(t)$. The maximum value in $N(\boldsymbol{x}(t))$, in practice approximated by some inner maximisation heuristic, is used as an estimate of the worst case cost $\tilde{g}(\boldsymbol{x}(t))$ at $\boldsymbol{x}(t)$.

Next we wish to identify a set of so called high cost points $H_{\sigma(t)}(\boldsymbol{x}(t))$ in $N(\boldsymbol{x}(t))$, containing the points with the largest objective values in the uncertainty neighbourhood of $\boldsymbol{x}(t)$, see Figure 1.5b. Using a threshold value $\sigma(t)$ that is dynamically adjusted as the algorithm iterates, $H_{\sigma(t)}(\boldsymbol{x}(t))$ is defined as:

$$H_{\sigma(t)}(\boldsymbol{x}(t)) := \{\boldsymbol{x}' \in H \cap N(\boldsymbol{x}(t)) \mid f(\boldsymbol{x}') \geq \tilde{g}(\boldsymbol{x}(t)) - \sigma(t)\}$$

For $t = 0$, $\sigma(t)$ is set to:

$$\sigma(0) = \sigma_{init} * (\tilde{g}(\boldsymbol{x}(0)) - f(\boldsymbol{x}(0)))$$

[BNT10b] set σ_{init} to 0.2. In subsequent steps $\sigma(t)$ uses the final value of $\sigma(t-1)$.

It is from the next stage of the algorithm that the descent directions approach gets its name, as the aim is to identify a descent direction vector $\boldsymbol{d}$ which optimally points away from all points in $H_{\sigma(t)}(\boldsymbol{x}(t))$, see Figure 1.5c. A mathematical programming approach is used to maximise the angle θ between the vectors connecting the points in $H_{\sigma(t)}(\boldsymbol{x}(t))$ to the current candidate $\boldsymbol{x}(t)$, and $\boldsymbol{d}$. This is achieved using the following second order cone problem (SOCP):

$$\min_{\boldsymbol{d},\beta} \beta \tag{Soc}$$

$$\text{s.t. } \|\boldsymbol{d}\| \leq 1 \tag{Con1}$$

$$\boldsymbol{d}^T \left(\frac{\boldsymbol{h} - \boldsymbol{x}(t)}{\|\boldsymbol{h} - \boldsymbol{x}(t)\|} \right) \leq \beta \qquad \forall \boldsymbol{h} \in H_{\sigma(t)}(\boldsymbol{x}(t)) \tag{Con2}$$

$$\beta \leq -\varepsilon \tag{Con3}$$

Here ε is a small positive scalar, so from (Con3) β is negative. The left hand side of constraint (Con2) is the multiplication of $\cos\theta$ and $\|\boldsymbol{d}\|$, for all hcps in $H_{\sigma(t)}(\boldsymbol{x}(t))$ and a feasible direction $\boldsymbol{d}$. (Con2) therefore relates β to the maximum value for $\cos\theta$ across all hcps. As the objective (Soc) is to minimise β, and β is negative, the angle θ will be greater than 90^o and maximised. Also minimising β in combination with (Con1) normalises $\boldsymbol{d}$. A standard solver such as CPLEX can be used to solve this SOCP. When an optimal direction cannot be found, that is the SOCP cannot be solved, the algorithm stops: a local robust minimum has been reached.

Given a set of hcps the attempt to solve the SOCP is deterministic, however if no such solution can be found the algorithm does not simply stop on the assumption that no such point exists. The current candidate $\boldsymbol{x}(t)$ might actually be surrounded by hcps. The points in $H_{\sigma(t)}(\boldsymbol{x}(t))$ are dependent on the definition of 'high cost': $f(\boldsymbol{x}') \geq \tilde{g}(\boldsymbol{x}(t)) - \sigma(t)$, and it may be the case that the classification of hcps is too generous. This can be corrected by adapting $H_{\sigma(t)}(\boldsymbol{x}(t))$ through the setting of the $\sigma(t)$ value. [BNT10b] suggest that when the SOCP is infeasible $\sigma(t)$ is reduced by dividing it by a factor α, which [BNT10b] set to 1.05. This increases the threshold for membership of $H_{\sigma(t)}(\boldsymbol{x}(t))$. The reduction of $\sigma(t)$, re-determination of $H_{\sigma(t)}(\boldsymbol{x}(t))$, and attempt to solve the SOCP can be repeated multiple times, up to some lower threshold for $\sigma(t)$, σ_α which [BNT10b] set to 0.001. The terminating criteria for the algorithm is when the SOCP is infeasible and $\sigma(t)$ is less than σ_α. Only at this stage it is assumed that a local robust minimum has been reached.

The final component of the algorithm is the calculation of the size of step to be taken in the descent direction $\boldsymbol{d}$, see Figure 1.5d. A step size $\rho_*(t)$ just large enough to ensure that all of the points in $H_{\sigma(t)}(\boldsymbol{x}(t))$ are at least on the boundary of the Γ-uncertainty neighbourhood of the next candidate solution is used: $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \rho_*(t) \cdot \boldsymbol{d}$. Where

$$\rho_*(t) = \min_{\rho(t)} \rho(t),$$

for:

$$\rho(t) = \boldsymbol{d}^T(\boldsymbol{h} - \boldsymbol{x}(t)) + \sqrt{(\boldsymbol{d}^T(\boldsymbol{h} - \boldsymbol{x}(t)))^2 - \|\boldsymbol{h} - \boldsymbol{x}(t)\|^2 + \Gamma^2}, \ \forall \boldsymbol{h} \in H_{\sigma(t)}(\boldsymbol{x}(t)).$$

This can be solved by simply evaluating over all members of $H_{\sigma(t)}(\boldsymbol{x}(t))$. In order to ensure that the heuristic makes reasonable progress with every step t, a minimum step size $\rho_{min}(t)$ is enforced. Nominally $\rho_{min}(0)$ is set to $\Gamma \cdot 0.01$, with $\rho_{min}(t)$ further decreasing with every step t by multiplication with a factor ρ_{red}: $\rho_{min}(t+1) = \rho_{min}(t) \cdot \rho_{red}$. A value of 0.99 is used for ρ_{red} in [BNT10b].

However prior to finalising the step to be taken and moving to the next candidate point: $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \rho_*(t) \cdot \boldsymbol{d}$, one final direction-distance check is performed. The uncertainty neighbourhood high cost set is temporarily extended by increasing the candidate's neighbourhood radius. The step size to be taken is added to Γ:

$$H_{\sigma(t)}(\boldsymbol{x}(t))_{updated} := \{\boldsymbol{x}' \mid \boldsymbol{x}' \in H \ , \ \|\boldsymbol{x}' - \boldsymbol{x}(t)\| \leq \Gamma + \rho_*(t) \ , \ f(\boldsymbol{x}') \ \geq \ \tilde{g}(\boldsymbol{x}(t)) \ - \ \sigma(t)\}$$

Now a check is made to ensure that the descent direction $\boldsymbol{d}$ still points away from the modified high cost set $H_{\sigma(t)}(\boldsymbol{x}(t))_{updated}$. This is achieved by calculating the dot product of all points in $H_{\sigma(t)}(\boldsymbol{x}(t))_{updated}$ and $\boldsymbol{d}$. If all dot products are negative the descent direction points away from all points in $H_{\sigma(t)}(\boldsymbol{x}(t))_{updated}$. In this case the step to the next candidate point is taken and the search continues: $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \rho_*(t) \cdot \boldsymbol{d}$.

This additional check is an attempt to avoid the upcoming step to the next candidate point $\boldsymbol{x}(t+1)$ being a mis-step into a region containing undesirable hcps. If the check fails and $\boldsymbol{d}$ is deemed no longer a valid direction, the algorithm returns to the SOCP cycle, this time using $H_{\sigma(t)}(\boldsymbol{x}(t))_{updated}$ in place of $H_{\sigma(t)}(\boldsymbol{x}(t))$. This may involve further reductions in $\sigma(t)$, by dividing it by the factor α up to the threshold σ_{α}.

Due to the potentially multiple attempts to identify a descent direction $\boldsymbol{d}$ and to ensure that this direction is valid for points in the neighbourhood of the current candidate $\boldsymbol{x}(t)$, d.d. is quite comprehensive in its identification of the appropriate steps to take from an initial start point. For full details of the descent directions algorithm refer to [BNT10b].

The inner maximisation component of the min max search that is employed by d.d. in [BNT10b] involves $n+1$ two-stage gradient ascent searches within the Γ-uncertainty neighbourhood of a given candidate point. However such an approach is likely to be impractical in many real-world situations, both in the face of some budget on numbers of function evaluations and with increasing dimension n. In our work the outer minimisation d.d. search is fed by an inner maximisation consisting of either uniform random sampling in a Γ-radius hypersphere around each candidate point, or in the work described in Chapter 4 a choice of inner search methods is available.

1.3.5 Particle swarm optimisation

PSO is a population based approach which moves a 'swarm' of particles through points in the decision variable space, performing function evaluations and iterating particle positions through the use of particle level 'velocity' vectors [KE95, KES01, Tal09]. Velocities are based on particle histories, shared information from the swarm, scaling, and randomisation. The intention is for the behaviour of this complex systems of particles to approximate a global optimisation search of the solution space. A standard PSO can be extended to a brute force robust PSO (rPSO) through the addition of some form of inner maximisation at a particle level. As with d.d., in the first instance this is achieved in our work using uniform random sampling in a Γ-radius hypersphere around each candidate point, however in Chapter 4 this is extended to include explicit inner search methods.

In a basic non-robust PSO formulation, the swarm (population) of N particles start at iteration $t=0$ randomly located at points $\boldsymbol{x}^j(0)$ in $\mathcal{X}$, where the function is evaluated; here $j=1,\ldots,N$. Each particle stores information on the best position it has visited in its history, $\boldsymbol{x}_*^j$, where best refers to the lowest objective function value $\tilde{g}(\boldsymbol{x}_*^j)$.

Information sharing is a key element of PSO, with each particle associated with a neighbourhood of other particles. Within a neighbourhood information about the best point visited by any particle in the neighbourhood within their entire histories, $\hat{\boldsymbol{x}}_*$, is shared. Again best refers to the lowest objective function value $\tilde{g}(\hat{\boldsymbol{x}}_*)$. A variety of neighbourhood topologies are available [EK95, KE95, KM02, MKN03, JM05, dCBF09, MP17]. Possibly the simplest information sharing approach employs a single neighbourhood with all particles having access to the current global best location information.

A particle is moved to a location $\boldsymbol{x}^j(t)$ at iteration t, through the addition of that particle's current velocity vector $\boldsymbol{v}^j$ to its previous position:

$$\boldsymbol{x}^j(t) = \boldsymbol{x}^j(t-1) \ + \ \boldsymbol{v}^j(t) \tag{PSOmove}$$

Again there are a number of alternative velocity formulations. For example a common basic formulation includes a so-called inertia [SE98, KES01] coefficient:

$$\boldsymbol{v}^j(t) = \omega \cdot \boldsymbol{v}^j(t-1) \ + \ C_1 \cdot \boldsymbol{r}_1 \cdot (\boldsymbol{x}_*^j - \boldsymbol{x}^j(t-1)) \ + \ C_2 \cdot \boldsymbol{r}_2 \cdot (\hat{\boldsymbol{x}}_* - \boldsymbol{x}^j(t-1)) \tag{Inertia}$$

Here particle velocities $\boldsymbol{v}^j(0)$ are initialised by uniform random sampling $\sim U(0\ ,\ 0.1)^n$ [Eng12]. Each component of the random vectors $\boldsymbol{r}$ is typically randomly sampled individually, $\boldsymbol{r}_1\ ,\ \boldsymbol{r}_2 \sim U(0\ ,\ 1)^n$. Vector multiplication is component wise. The scalar terms C_1 and C_2 represent weightings that a particle puts on its $\boldsymbol{x}_*^j$ (C_1) versus $\hat{\boldsymbol{x}}_*$ (C_2) location data, whilst the inertia scalar ω moderates the significance of the preceding velocity.

As particles move through $\mathcal{X}$ their individual $\boldsymbol{x}_*^j$ values and the global $\hat{\boldsymbol{x}}_*$ are updated. If at any stage the next candidate position for any particle lies outside of $\mathcal{X}$, an invisible boundary condition is adopted [RR04] in the formulations employed here. Particles are allowed to leave the feasible region to naturally return to feasibility due to the pull of the $\boldsymbol{x}_*^j$ and $\hat{\boldsymbol{x}}_*$ information. Note that when a candidate moves outside of the feasible region no function evaluations are undertaken. Rather the velocity equation is updated by the particle's new location, with the $\boldsymbol{x}_*^j$ information remaining unchanged.

In its application here, when the budget of available function evaluations is exhausted the current global best $\hat{\boldsymbol{x}}_*$ location is accepted as the estimate of the position of the robust global minimum.

This completes the literature review. In Section 1.4 we give a brief overview of the development of novel metaheuristics for robust problems as described in Chapters 2 to 4, including the current status of publications and statements of authorship.

1.4 Description and status of papers

1.4.1 Published and submitted papers

Chapters 2 to 4 comprise the following papers:

- A Largest Empty hypersphere metaheuristic for robust optimisation with implementation uncertainty [HGW19]. This was published in the journal *Computers & Operations Research* in 2019.

A novel global metaheuristic for robust problems is developed and tested on a suite of multi-dimensional test problems. The new technique, the largest empty hypersphere approach extends the d.d. idea of moving away from identified high cost points locally, to a global perspective. This is an individual based approach which starts from some random candidate point in the decision variable space where a neighbourhood uncertainty analysis is performed. All function evaluations are recorded in a history set. A high cost threshold equal to the current estimate of the robust global minimum is used to differentiate hcps in the history set from all other points. Given the set of hcps the largest region in the feasible solution space that contains no hcps is identified. The search then moves to the centre of this empty region. This is the new candidate point. This repeats until no region empty of hcps can be identified or some other limiting factor is reached, e.g. the budget of function evaluations is exceeded.

Several approaches to identifying the largest empty hypersphere (LEH) devoid of all hcps are considered, and it is established that the use of a genetic algorithm is most appropriate. A suite of eight multi-dimensional test problems, employed across five dimensions between 2D and 100D, are used to test the effectiveness of LEH against a brute force robust PSO (rPSO) and a re-starting d.d.. In all cases the inner maximisation is by random sampling, and a budget of 10,000 function evaluations is used. Parameter tuning is applied to each heuristic. LEH is shown to outperform the comparator techniques, particularly for problems of higher dimension.

- Particle Swarm Metaheuristics for Robust Optimisation with Implementation Uncertainty [HGD20b]. This was accepted for publication in the journal *Computers & Operations Research* in May 2020.

Here a new robust metaheuristic framework is developed encompassing two new capabilities which can be used in combination or individually. Building on the d.d. and LEH approaches, a population based approach is developed which incorporates elements of both local exploitation and global exploration. From the d.d. approach we take the notion of calculating a local vector pointing away from the worst points within the Γ-uncertainty neighbourhood of a candidate point. In a PSO setting we use the calculation of a descent direction vector at an individual particle's candidate location and add it to the baseline velocity formulation (Inertia), weighted by a scalar C_3 term and randomised vector $\boldsymbol{r}_3$.

From LEH we take the concept of a stopping condition and relocation to the centre of the largest hypersphere devoid of hcps. The stopping condition terminates an inner maximisation search prematurely if it is determined that the robust value of the candidate cannot improve on the current estimate of the robust global minimum. This has the potential for significant efficiency savings in terms of function evaluations, enabling greater exploration. In a PSO setting it is recognised that this can be applied at a particle level using particle best information for the stopping threshold. This can also be taken a step further.

In a standard LEH setting candidate points will not be located near hcps. However in a PSO setting a particle may move to a point such that its Γ-uncertainty neighbourhood already contains hcps, relative to that particle. In such a situation no function evaluations are required. This introduces the concept of 'dormancy' whereby individual particles may repeatedly require no function evaluations over multiple iterations of the swarm. A second form of dormancy, where particles repeatedly move outside of the feasible region is also considered. In either case, after some number of dormant iterations a new capability is introduced, that of relocating the dormant particle to the centre of the LEH devoid of all hcps greater than the current estimate of robust global minimum.

Ten multi-dimensional test problems are used over six dimensions between 2D and 100D, to test the effectiveness of the new framework employed in its three settings: an rPSO + d.d. capability, an rPSO + LEH capability, and an rPSO + d.d. + LEH capabilities. A brute force rPSO and a re-starting d.d. are used as comparators, along with LEH. In all cases inner maximisation is by random sampling, a budget of 5,000 function evaluations is applied, and parameters are tuned for each heuristic at each dimension. The new framework is shown to outperform the comparator techniques, although LEH also performs well at 100D.

- Automatic Generation of Algorithms for Robust Optimisation Problems using Grammar-Guided Genetic Programming [HGD20a]. This was submitted to the journal *Applied Soft Computing* in April 2020.

Here new metaheuristics for robust problems are automatically generated using genetic programming (GP). From an initial random population of heuristics, new generations of heuristics are evolved. On completion of the GP process the fittest is deemed the best heuristic. This work requires the generation of a number of sub-algorithmic building blocks from which complete heuristics can be constructed, along with the design rules by which they are combined. This is our grammar. In addition we must generate an evolutionary GP framework which can operate on heuristics constructed from the sub-components. By generating a context free grammar and using a common GP representation of an algorithm (heuristic) as a tree, we are able to use standard tree-based combination and mutation operators within the GP process.

Here all heuristics are based on an rPSO formulation, but a number of different network topologies, movement formulations based on elements of the LEH and d.d. capabilities developed in [HGD20b], inner maximisation search techniques, and other features are available in component form within the grammar.

A series of GP runs are conducted using an experimental test suite comprising ten multi-dimensional test problems across two dimensions, 30D and 100D. Ten separate GP runs are undertaken, one for each test problem individually, plus one combined run using all ten problems simultaneously. This is repeated for 30D and 100D. A budget of 2,000 function evaluations is assumed for all runs.

For the general (ten case) best performing heuristics identified in the GP analysis, a direct comparison can be made against the best performing heuristics from our paper [HGD20b]. This is notwithstanding the use in the GP runs of a budget of just 40% of that used in the comparator results. There is a strong improved performance by the best new general heuristics, including substantial outperformance for some problems. For the individual case results the assessment against the comparators can be taken as indicative only, as the comparators were not specifically designed to tackle individual problems. Nevertheless the improved performance of the new best heuristics for individual test cases is encouraging.

In addition the population of heuristics generated by each GP run give us a large number of heuristics on which we can undertake some assessment of heuristic component breakdown against performance. The features which can be associated with good performance include inner maximisation by random sampling on a small number of points, particle level stopping conditions, a small swarm size, a Global topology, and particle movement using an inertia formulation plus LEH and d.d. components.

- In addition to the three papers [HGW19, HGD20b, HGD20a] I am a co-author of the paper: Representative scenario construction and preprocessing for robust combinatorial optimization problems [GH19]. This was published in the journal *Optimization Letters* in 2019.

This work employs a genetic algorithm to evolve heuristics for robust problems, specifically to optimise combinatorial problems. The robust approaches developed and analysed are mathematical programs targeted at a specific form of problem. Due to the use of a fairly simple grammar of components and design rules, a GA is used here as opposed to a more complex evolutionary approach such as GP.

My contribution relates to the conceptualisation of employing an automatic generation of algorithms approach, including the background work on how this could be achieved and writing relevant sections of text. I also undertook some document reviews.

1.4.2 Statement of authorship

The following describes my contribution as lead author on [HGW19, HGD20b, HGD20a], and the contributions of the co-authors:

- My contribution is consistent across all three papers:
  - Conceptualization: Formulating and developing research goals and aims.
  - Methodology: Development of the methods employed and analysis undertaken.
  - Designing and developing the algorithms in Java, including the evolutionary tuning and genetic programming frameworks. Also developing the R code for conducting analysis.
  - Performing experimental runs.
  - Testing, verification and validation of code and the results of experimental testing.

  - Application of statistical techniques, and other forms of analysis to the results of the experimental testing.
  - Preparation and writing of the text of the papers, including responding to reviewer's comments and undertaking amendments.
  - Preparation and generation of the tables and figures.

- Marc Goerigk: Marc's contribution is consistent across all three papers:
  - Supervising the research, including planning and execution.
  - Conceptualization: Formulating and developing research goals and aims.
  - Methodology: Development of the methods employed and analysis undertaken.
  - Performing experimental runs.
  - Reviewing and editing the paper, including critical review and commentary.

- Michael Wright: Mike's contribution to the paper [HGW19] is:
  - Supervising the research, including planning and execution.
  - Reviewing and editing the paper, including critical review and commentary.

- Trivikram Dokka: Vikram's contribution to the papers [HGD20b, HGD20a] is:
  - Supervising the research, including planning and execution.
  - Reviewing and editing the paper, including critical review and commentary.

Chapter 2

Paper 1: A Largest Empty Hypersphere Metaheuristic for Robust Optimisation with Implementation Uncertainty

Published in *Computers & Operations Research* in 2019 [HGW19].
Partially funded through EPSRC grants EP/L504804/1 and EP/M506369/1.

Author 1: Martin Hughes, Lancaster University, United Kingdom.
Author 2: Marc Goerigk, University of Siegen, Germany.
Author 3: Michael Wright, Lancaster University, United Kingdom.

Abstract: We consider box-constrained robust optimisation problems with implementation uncertainty. In this setting, the solution that a decision maker wants to implement may become perturbed. The aim is to find a solution that optimises the worst possible performance over all possible perturbances.

Previously, only few generic search methods have been developed for this setting. We introduce a new approach for a global search, based on placing a largest empty hypersphere. We do not assume any knowledge on the structure of the original objective function, making this approach also viable for simulation-optimisation settings. In computational experiments we demonstrate a strong performance of our approach in comparison with state-of-the-art methods, which makes it possible to solve even high-dimensional problems.

2.1 Introduction

The use of models to support informed decision making is ubiquitous. However, the size and nature of the decision variable solution space, and the model run time, may make a comprehensive – exhaustive or simply extensive – evaluation of the problem space computationally infeasible. In such cases an efficient approach is required to search for global optima.

Mathematical programs are one form of model that are explicitly formulated as optimisation problems, where the model representation imposes assumptions on the structure of the decision variable space and objective function. Such models are well suited to efficient solution, and identification of global optima may be theoretically guaranteed when feasible solutions exist. However many real-world problems are not suited to expression as a mathematical program (e.g., a solution is evaluated by using a simulation tool). From an optimisation perspective models where no assumptions are made about the model structure can be thought of as a black-box, where decision variables values are input and outputs generated for interpretation as an objective. In this case optimisation search techniques such as metaheuristics are required, i.e., general rule-based search techniques that can be applied to any model.

An additional widespread feature of many real-world problems is the consideration of uncertainty which may impact on model outputs, and so on corresponding objective function values. One strategy is to simply ignore any uncertainty and perform a standard search, possibly assessing and reporting on the sensitivity of the optimum after it has been identified. However it has been established that optimal solutions which are sensitive to parameter variations within known bounds of uncertainty may substantially degrade the optimum objective function value, meaning that solutions sought without explicitly taking account of uncertainty are susceptible to significant sub-optimality, see [BTEGN09, GS16]. In the face of uncertainty the focus of attention for an optimisation analysis shifts from the identification of a solution that just performs well in the expected case, to a solution that performs well over a range of scenarios.

In this paper we develop a new algorithm for box-constrained robust black-box global optimisation problems taking account of implementation uncertainty, i.e., the solution that a decision maker wants to implement may be slightly perturbed in practice, and the aim is to find a solution that performs best under the worst case perturbation. Our method is based on an exploration technique that uses largest empty hyperspheres (LEHs) to identify regions that can still contain improving robust solutions. In a computational study we compare our method with a local search approach from the literature (see [BNT10b]) and a standard particle swarm approach. We find that our approach considerably outperforms these methods, especially for higher-dimensional problems.

Structure of this paper. We begin with a review of the literature on metaheuristics for robust optimisation in Section 2.2 before outlining the formal description of robust min max problems in Section 2.3. We also consider some of the details of the established local robust search technique due to [BNT10b]. In Section 2.4 we introduce a novel approach, an exploration-focused movement through the search space identifying areas that are free of previously identified poor points. We include a discussion and descriptions of the algorithms used to identify empty regions of the decision variable search space. The approach is then tested against alternative heuristics in Section 2.5, on test problems of varying dimension. The experimental set up is described and the results of this analysis presented. Finally we summarise and consider further extensions of this work in Section 2.6.

2.2 Literature review

2.2.1 Robust optimisation

Different approaches to model uncertainty in decision making problems have been explored in the literature. Within robust optimisation, a frequent distinction is made between parameter uncertainty (where the problem data is not known exactly) and implementation uncertainty (where a decision cannot be put into practice with full accuracy). Implementation uncertainty is also known as decision uncertainty [BTEGN09, Tal09, BNT10b].

A common approach to the incorporation of uncertainty for black-box problems is stochastic optimisation. Here knowledge of the probability distributions of the uncertain parameters is assumed and some statistical measure of the fitness of a solution assessed, e.g. using Monte Carlo simulation to estimate the expected fitness. This may be the expected value, or a more elaborate model such as the variance in the fitness of a solution, or even a multi-objective optimisation setting, see [PBJ06, HdMB14].

An alternative to a stochastic approach is robust optimisation, whose modern form was first developed in [KY97] and [BTN98]. Whereas with stochastic optimisation a knowledge of probability distributions over all possible scenarios is typically assumed, in robust optimisation it is only assumed that some set is identified containing all possible uncertainty scenarios (potentially infinite in number). A classic robust approach is then to find a solution across all scenarios that is always feasible (strictly robust) and optimises its performance in the worst case. This is known as min max. For a given point in the decision variable space there is an 'inner' objective to identify the maximal (worst case) function value in the local uncertainty neighbourhood, and an overall 'outer' objective to identify the minimum such maximal value.

The field of robust optimisation has been primarily aligned with mathematical programming approaches. There the methodology is based around the definition of reasonable uncertainty sets and the reformulation of computationally tractable mathematical programming problems. For specific forms of convex optimisation problems, the problem incorporating uncertainty can be re-formulated to another tractable, convex problem, see [BNT07, GS10]. To overcome concerns that the strictly robust worst case approach may be overly conservative, the concept of robustness can be expanded in terms of both the uncertainty set considered and the robustness measure [GS16]. On the assumption that it is overly pessimistic to assume that all implementation errors take their worst value simultaneously [BS04] consider an approach where the uncertainty set is reduced, and a robust model defined where the optimal solution is required to remain feasible for uncertainty applied to only a subset of the decision variables at any given time. Min max regret, see [ABV09], is an alternative to min max, seeking to minimise the maximum deviation between the value of the solution and the optimal value of a scenario, over all scenarios. [BTBB10] considers soft robustness, which utilises a nested family of uncertainty sets. The distributionally robust optimisation approach, see [GS10], attempts to bridge robust and stochastic techniques by utilizing uncertainty defined as a family of probability distributions, seeking optimal solutions for the worst case probability distribution. [CG16] use a bi-objective approach to balance average and worst case performance by simultaneously optimising both.

Robust optimisation in a mathematical programming context has been application-driven, so considerable work has been undertaken in applying robustness techniques to specific problems or formulations, see [BS07, GS16]. There has also been some cross-over into the application of specific heuristics, for example see [GLT97, AVCMSdCM11]. However application to general problems has been less well addressed [GS16]. Furthermore robust approaches applied to black-box models are much less widely considered than approaches for mathematical programming problems, see [MWPL13, GS16, MWPL16]. Recently, robust optimisation with implementation uncertainty has also been extended to multi-objective optimisation, see [EKS17].

2.2.2 Metaheuristic for robust optimisation

The min max approach has been tackled with standard metaheuristic techniques applied to both the inner maximisation and outer minimisation problems. In co-evolutionary approaches two populations (or swarms) evolve separately but are linked. The fitness of individuals in one group is informed by the performance of individuals in the other, see [CSZ09]. [Her99, Jen04] use such a two-population genetic algorithm (GA) approach, whilst [SK02, MKA11] consider two-swarm co-evolutionary particle swarm optimisation (PSO) techniques for min max problems. A brute force co-evolutionary approach is to employ complete inner maximisation searches to generate robust values for each individual in each generation of the outer minimisation, however this is expensive in terms of model runs (i.e., function evaluations), see [MWPL16]. More practical co-evolutionary approaches, for example using only small numbers of populations for the outer search and the inner (uncertainty) search which share information between populations from generation to generation, or following several generations, require the application of additional simplifications and assumptions, see [CSZ09, MKA11].

One general area of research is the use of emulation to reduce the potential burden of computational run times and the number of model-function evaluations, see [KVDHL16]. [ZZ10] use a surrogate-assisted evolutionary algorithm to tackle the inner search for black-box min max problems. [MWPL13, MWPL16] employs Kriging meta-modelling coupled with an expected improvement (EI) metric, as well as a relaxation of the inner maximisation search. The EI metric is used to efficiently choose points in the decision variable space where nominal (expensive) function evaluation should be undertaken, see [JSW98], here with a view to most efficiently improving the estimate of the robust global minimum. The relaxation involves iteratively performing the outer minimisation on a limited inner uncertainty neighbourhood followed by an extensive inner maximisation search in the region of the identified outer minimum. This continues whilst the inner search sufficiently deteriorates the outer solution, with the inner maximum point being added to the limited inner uncertainty set with each iteration.

A second approach due to [uRLvK14, uRL17] also uses Kriging and an EI metric, building on a meta-model of the expensive nominal problem by applying a robust analysis directly to the Kriging model and exploiting the fact that many more inexpensive function evaluations can be performed on this meta-model. A modified EI metric is calculated for the worst case cost function of the meta-model, to efficiently guide the search in the nominal expensive function space. In [uRL17] the approach is applied to a constrained non-convex 2 dimensional problem due to [BNT10b, BNT10a], the unconstrained version of which is also considered here. The Kriging-based approach is shown to significantly outperform the approaches outlined here, in terms of the number of expensive function evaluations required to converge towards the robust optimum. In general we would expect the approach from [uRLvK14, uRL17] to outperform the approaches considered here, in terms of efficiency when applied to low dimensional non-convex problems. However the primary challenge with meta-model based approaches is their application to higher dimensional problems. The test cases considered in [MWPL13, MWPL16, uRLvK14, uRL17] have either been restricted to low dimensional non-convex problems, or simpler convex and convex-concave problems of up to 10 dimensions.

One local black-box min max approach is due to [BNT07, BNT10b, BNT10a]. Here a search is undertaken by iteratively moving along 'descent directions'. Uncertainty around individual points is assessed using local gradient ascents, based on which undesirable 'high cost points' (hcps) are identified. Steps are taken in directions which point away from these hcps, until no direction can be found.

Our approach is inspired by both elements of the descent directions technique and the concept of relaxation of the inner maximisation search. We extend the idea of locally moving away from identified hcps to a global perspective, seeking regions of the solution space currently empty of such undesirable points. Furthermore the nature of our outer approach enables the curtailing of an inner maximisation search if it is determined that the current point under consideration cannot improve on the current best robust global solution.

2.3 Notation and previous results

2.3.1 Problem description

We consider a general optimisation problem of the form

$$\begin{aligned} &\min f(\boldsymbol{x}) \\ &\text{s.t. } \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ denotes the n-dimensional vector of decision variables, $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, and $\mathcal{X} \subseteq \mathbb{R}^n$ is the set of feasible solutions. We write $[n] := \{1, \ldots, n\}$. In this paper, we assume box constraints $\mathcal{X} = \prod_{i \in [n]} [l_i, u_i]$. Any other potential feasibility constraints are assumed to be ensured through a penalty in the objective.

In implementation uncertainty, we assume that a desired solution $\boldsymbol{x}$ might not be possible to put into practice with full accuracy. Instead, a "close" solution $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \Delta\boldsymbol{x}$ may be realised. The aim is to find a robust $\boldsymbol{x}$ such that for any such solution $\tilde{\boldsymbol{x}}$ from the neighbourhood of $\boldsymbol{x}$, the worst case performance is optimised.

More formally, we follow the setting of [BNT10b] and consider the so-called uncertainty set

$$\mathcal{U} := \{\Delta\boldsymbol{x} \in \mathbb{R}^n \mid \|\Delta\boldsymbol{x}\| \leq \Gamma\}$$

where $\Gamma > 0$ defines the magnitude of the uncertainty, and $\|\cdot\|$ refers to the Euclidean norm. The worst case costs of a solution $\boldsymbol{x} \in \mathcal{X}$ are then given as

$$g(\boldsymbol{x}) := \max_{\Delta\boldsymbol{x} \in \mathcal{U}} f(\boldsymbol{x} + \Delta\boldsymbol{x})$$

and so the robust optimisation problem is given by:

$$\min_{\boldsymbol{x} \in \mathcal{X}} g(\boldsymbol{x}) = \min_{\boldsymbol{x} \in \mathcal{X}} \max_{\Delta\boldsymbol{x} \in \mathcal{U}} f(\boldsymbol{x} + \Delta\boldsymbol{x}) \tag{ROP}$$

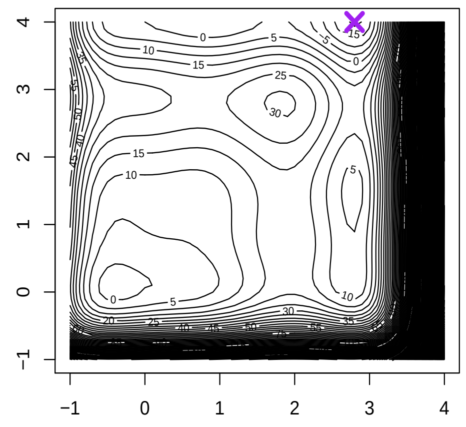
We therefore have an inner maximisation and outer minimisation problem, such that the identification of the robust global optimum is based on finding the (outer) minimum worst case cost objective function value in the decision variable space, and that objective is determined by the (inner) maximisation of the nominal objective function in the uncertainty neighbourhood around each point in the decision variable space. This type of problem is also known as min max.

Note that $\boldsymbol{x} + \Delta\boldsymbol{x}$ may not be in $\mathcal{X}$, for which reason we assume that the definition of f is not limited to $\mathcal{X}$. However, if it is desired that $\boldsymbol{x} + \Delta\boldsymbol{x} \in \mathcal{X}$ for all $\Delta\boldsymbol{x} \in \mathcal{U}$, then this can be ensured by reducing the size of the feasible search space by Γ.

As an example for our problem setting, consider the 2-dimensional polynomial function due to [BNT10b]:

$$\begin{aligned} f(x,y) =& 2x^6 - 12.2x^5 + 21.2x^4 + 6.2x - 6.4x^3 - 4.7x^2 - y^6 \\ & -11y^5 + 43.3y^4 - 10y - 74.8y^3 + 56.9y^2 - 4.1xy \\ & -0.1y^2x^2 + 0.4y^2x + 0.4x^2y \end{aligned} \tag{poly2D}$$

For a feasible solution space within bounds $[-1, 4]$ in each dimension, and uncertainty defined by a Γ-radius value of 0.5, the nominal and worst case plots for (poly2D) are shown in Figure 2.1. In min max the problem is one of finding the global minimum for the worst case cost function. If uncertainty is ignored the problem is just one of finding the global minimum of the (nominal) objective as shown in Figure 2.1a, whereas including uncertainty the problem becomes one of finding the (worst case cost) objective as shown in Figure 2.1b. In both cases the search proceeds based on generating nominal objective values but for the worst case cost we must further undertake some assessment of the impact of uncertainty on those objective outputs.


(a) Nominal problem

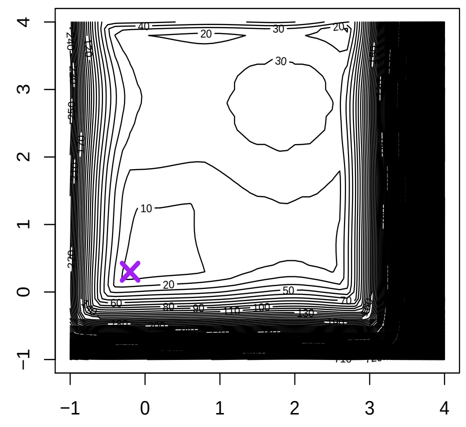
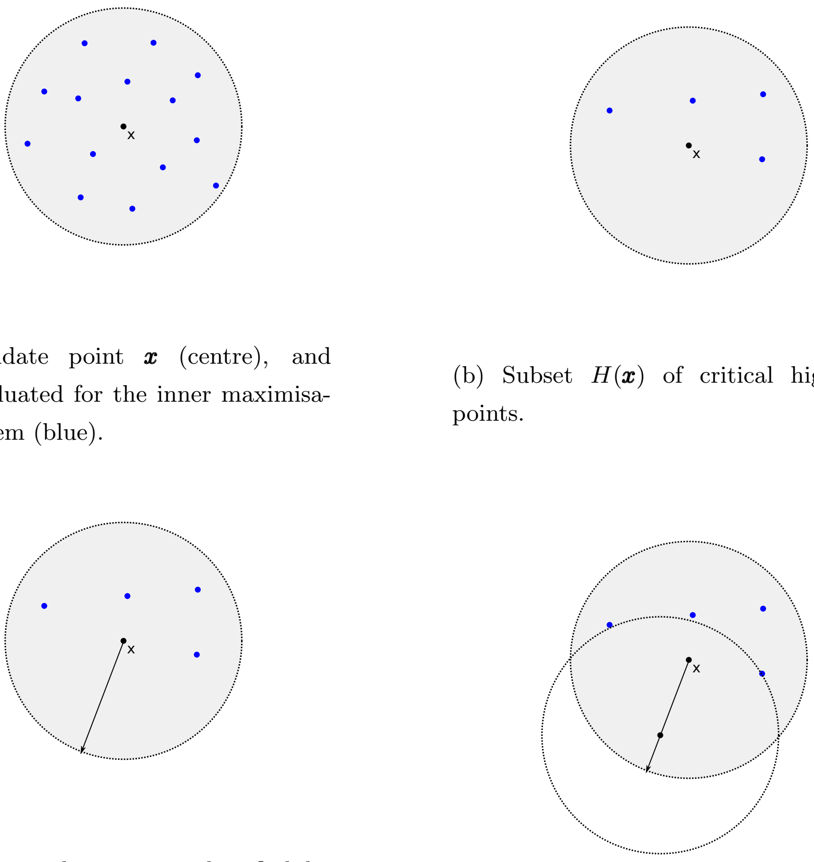

(b) Worst case problem with Γ=0.5

Figure 2.1: Nominal and worst case cost for (poly2D) from [BNT10b]. Marked in purple are the respective optima.

Here the global optimum value for the nominal problem is -20.8 at (2.8, 4.0). The worst case plot is estimated by randomly sampling large numbers of points in the Γ-uncertainty neighbourhood around each plotted point. The worst case cost at each point is then approximated as the highest value of $f(x)$ identified within each Γ-uncertainty neighbourhood. The global optimum for the worst case problem is approximately 4.3 at (-0.18, 0.29). The significant shift in the nominal versus robust optima, both in terms of its location and the optimum objective, emphasises the potential impact of considering uncertainty in decision variable values. The difference between the nominal and robust optimal objective function values is the 'price of robustness', see [BS04].

2.3.2 Local robust search using descent directions

We briefly summarise the local search approach for (ROP) that was developed in [BNT10b]. Here, (ROP) is solved using a local robust optimisation heuristic illustrated by Figure 2.2. An initial decision variable vector $\hat{\boldsymbol{x}}$ is randomly sampled. Then a series of gradient ascent searches are undertaken within the Γ-uncertainty neighbourhood of this candidate solution to identify hcps, see Figure 2.2a. This approximates the inner maximisation problem $\max_{\Delta \boldsymbol{x}} f(\hat{\boldsymbol{x}} + \Delta \boldsymbol{x})$. Using a threshold value that is dynamically adjusted during the algorithm, a subset $H(\hat{\boldsymbol{x}})$ of all evaluated points is identified, see Figure 2.2b.



(a) Candidate point $\boldsymbol{x}$ (centre), and points evaluated for the inner maximisation problem (blue).

(b) Subset $H(\boldsymbol{x})$ of critical high cost points.

(c) A descent direction is identified by solving a second order cone problem.

(d) The step size is determined.

Figure 2.2: Description of the descent direction robust local search approach [BNT10b].

In the next step, a descent direction is identified that points away from the set $H(\hat{\boldsymbol{x}})$, see Figure 2.2c. To this end, a mathematical programming approach is used, minimising the angle between the hcps and the candidate solution. This leads to the following second order cone problem.

$$\min_{\boldsymbol{d},\beta} \beta \tag{2.1}$$

$$\text{s.t. } \|\boldsymbol{d}\| \leq 1 \tag{2.2}$$

$$\boldsymbol{d}^T\boldsymbol{h} \leq \beta \qquad \forall \boldsymbol{h} \in H(\hat{\boldsymbol{x}}) \tag{2.3}$$

$$\beta \leq -\varepsilon \tag{2.4}$$

Here, $\boldsymbol{d}$ is the descent direction, which is normalised by Constraint (2.2). Constraints (2.3) ensure that β is the maximum angle between $\boldsymbol{d}$ and all high cost points $\boldsymbol{h}$. Through Constraint (2.4), we require a feasible descent direction to point away from all points in $H(\hat{\boldsymbol{x}})$. When an optimal direction cannot be found, the algorithm stops – a robust minimum has been reached.

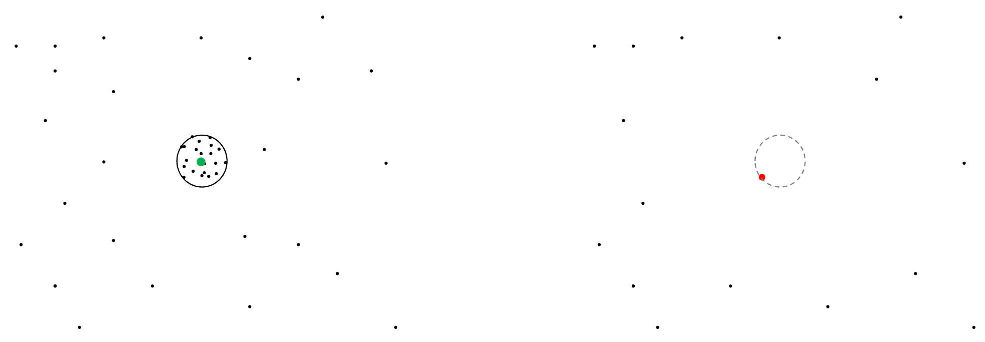
Next the size of the step to be taken is calculated, see Figure 2.2d. A step size just large enough to ensure that all of the hcps are outside of the Γ-uncertainty neighbourhood of the next candidate solution is used. Using the identified descent direction and step size the algorithm moves to a new candidate point, and so the heuristic repeats iteratively until a robust minimum has been identified.

2.4 A new largest empty hypersphere approach

2.4.1 Algorithm overview

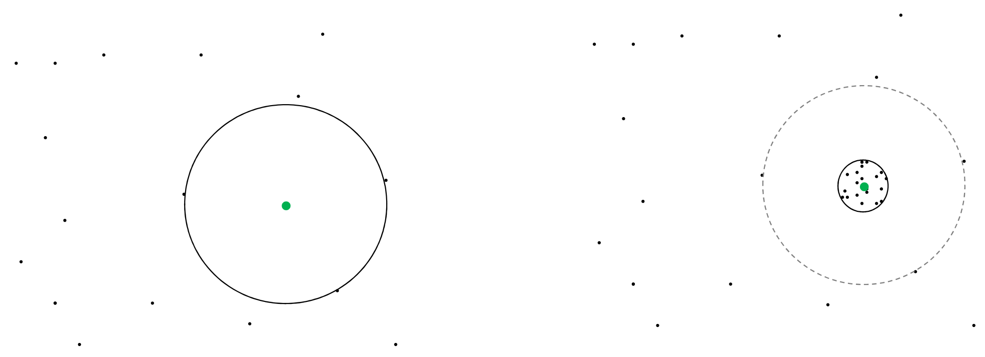
Building on the notion of a search that progresses locally by moving away from already identified poor (high cost) points, we develop a global approach that iteratively moves to the region of the decision variable solution space furthest away from recognised hcps. This is an exploration-focused approach, although rather than concentrating on examining unvisited regions the intention here is to identify and visit regions devoid of hcps. Assuming uncertainty as considered previously in terms of a single value Γ that defines a radius of uncertainty in all decision variables, we associate the idea of the largest empty region (empty of hcps) with the idea of the largest empty hypersphere (LEH), or largest empty circle in 2D. The approach is then to locate the next point in the search at the centre of the identified LEH, and to iteratively repeat this as more regions are visited and hcps identified. The approach is described in Figure 2.3.

We start by randomly sampling one or more points and evaluating the objective function f at each. From these start points a candidate point is selected and an inner analysis undertaken in the candidate's Γ-uncertainty neighbourhood with a view to identifying the local maximum, Figure 2.3a. This local worst case cost for the candidate is the first estimate of a robust global minimum, that is a global min max, and is located at the candidate point. The aim is now to move to a point whose uncertainty neighbourhood has a lower worst case cost than the current global value. We seek to achieve this by identifying the largest hypersphere of radius at least Γ within the defined



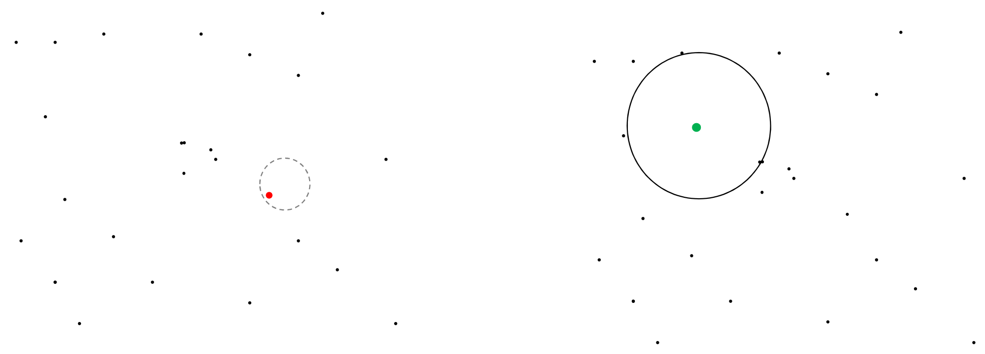
(a) The decision variable space is seeded randomly. Perform an inner search around one candidate point.

(b) The current high cost set, including one point from the previous inner search and some of the seed points.



(c) Identify the largest empty hypersphere, the centre of which is the next candidate point.

(d) Inner search around the new candidate. The robust value here is less than the current global minimum.



(e) The current high cost set, including more previously evaluated points due to the reduced high cost threshold.

(f) Identify the largest empty hypersphere, the centre of which is the next candidate point.

Figure 2.3: Description of largest empty hypersphere (LEH) approach.

feasibility bounds which is completely empty of hcps, and moving to the centre of that LEH, see Figures 2.3b - 2.3c.

All points evaluated are recorded in a history set, a subset of which forms the high cost set. The high cost set contains a record of all points evaluated so far with an objective function value greater or equal to a high cost threshold, and here the high cost threshold is set as the current estimate of the robust global minimum. Both the history set and the high cost set are updated as more points are visited and the high cost threshold reduces, see Figures 2.3d - 2.3e. On performing all inner searches after the first candidate, a candidate's robust value may be no better than the current estimate of the robust global minimum (and therefore the current high cost threshold), in which case at least one point will be added to the high cost set. Alternatively if a candidate's robust value is better than the current estimate of the robust global minimum, this current recorded optimum is overwritten and the high cost threshold reduced accordingly. Again this introduces at least one high cost point to the high cost set, but the reducing threshold may also introduce additional points from the history set; this is suggested in Figure 2.3e.

The search stops when no LEH of radius greater than Γ exists or some pre-defined resource limit has been reached. Then the candidate point around which the current estimate of the robust global minimum has been determined is deemed the robust global minimum. Otherwise the search repeats, performing analysis in the Γ-uncertainty neighbourhood around candidates to estimate the local (inner) max, updating the global minimum worst case cost if appropriate, and moving to the next identified LEH, Figure 2.3f.

The critical feature of such an approach is the identification of regions of the solution space that are currently empty of, and furthest from, the undesirable hcps. As defined here this corresponds to identifying the largest hypersphere devoid of all hcps.

Given a discrete history set H of all points evaluated so far, high cost points are those members of H with objective value which is at least the current high cost threshold τ, i.e.,

$$H_\tau := \{\boldsymbol{h} \in H \mid f(\boldsymbol{h}) \geq \tau\}$$

We denote $N_\tau = |H_\tau|$ as the cardinality of H_τ, and write $H_\tau = \{\boldsymbol{h}^1, \ldots, \boldsymbol{h}^{N_\tau}\}$. The identification of a point $\boldsymbol{p} \in \mathcal{X}$ which is furthest from all N_τ high cost points in H_τ is a max min problem:

$$\max_{\boldsymbol{p} \in \mathcal{X}} \min_{i \in [N_\tau]} d(\boldsymbol{p}, \boldsymbol{h}^i), \tag{LEHP}$$

where $d(\boldsymbol{p}, \boldsymbol{q})$ is the Euclidean distance between two points $\boldsymbol{p}$ and $\boldsymbol{q}$, see [OS97].

In the following, we specify this general LEH approach by considering two aspects in more detail: The *outer search* is concerned with placing the next candidate point $\boldsymbol{x}$ by solving (LEHP). The inner search then evaluates this candidate by calculating $g(\boldsymbol{x})$ approximately.

2.4.2 Outer search methods

Here we will introduce different approaches to identifying the largest empty hypersphere, given a set of high cost points H_τ. It should be noted that none of these approaches requires additional function evaluations, which is usually considered the limiting resource in black-box settings.

2.4.2.1 Randomly sampled LEH algorithm

A very simple approach is to generate potential candidates randomly within the feasible region, then determine whether they are more than Γ away from all hcps. If so they are a valid candidate, if not re-sample up to some defined maximum number of times beyond which it is assumed that no such candidate point can be found and the solution has converged on a robust global minimum. Rather than being a largest empty hypersphere approach this is just a valid empty hypersphere approach, and the size of the identified empty hypersphere might vary considerably from one candidate to the next.

2.4.2.2 Genetic Algorithm for LEH

The solution of (LEHP) is an optimisation problem. Furthermore, given a point $\boldsymbol{p}$ which is a potential candidate for the centre of the largest empty hypersphere, the inner minimisation calculation in (LEHP) involves just an enumeration over the N_τ Euclidean distance calculations between each hcp and $\boldsymbol{p}$ to identify the minimum distance $d(\boldsymbol{p}, \boldsymbol{h}^k)$, where $\boldsymbol{h}^k$ is the closest hcp. Therefore the focus for the solution of (LEHP) is the outer maximisation, for which we may consider an approximate heuristic approach. We employ a genetic algorithm (GA), a commonly cited evolutionary algorithm (EA) [Tal09]. Here each individual represents a point $\boldsymbol{p}$ in the decision variable space, and the objective function $f_{LEH}(\boldsymbol{p}) := \min_{\boldsymbol{h} \in H_\tau} d(\boldsymbol{p}, \boldsymbol{h})$ is the minimum distance between a given point $\boldsymbol{p}$ and all hcps in H_τ. We seek to maximise this minimal distance by evolving a population of points starting from randomly selected feasible points in the decision variable space $\mathcal{X}$. The best point generated by the GA is the next candidate point – that is estimated centre of the LEH, for the current H, τ and H_τ.

2.4.2.3 Voronoi based LEH

Within the literature a widely referenced approach for tackling low dimensional LEH problems is due to [Tou83], and is based on the geometric Voronoi diagram approach, see [Cha93, OS97]. The Voronoi approach partitions a space into regions (cells). For a given set of points each cell corresponds to a single point such that no point in the cell is closer to any other point in the set. Points on the edges between cells are equidistant between the set points which lie on either side of that edge. For our LEH problem the set of points is H_τ, and the Voronoi diagram approach corresponds to segmenting the feasible space $\mathcal{X}$ into $N_\mathcal{T}$ separate cells, one for each hcp. The (Voronoi) vertices that lie at the intersection of these cell (Voronoi) edges maximise the minimum distance to the nearby set points, see [Cha93, OS97]. So for a given $H_\mathcal{T}$ if we can determine the Voronoi diagram we can use the identified Voronoi vertices as potential candidate points $\boldsymbol{p}$. The solution of (LEHP) is then simply a matter of enumeration, for each $\boldsymbol{p}$ calculating the (inner) minimum Euclidean distance to all hcps, and then selecting the (outer) maximum such minimal distance.

The original approach due to [Tou83] includes the identification of vertices (candidate centres of LEHs) that can be sited outside of defined boundaries, in infeasible regions. This is not exactly as required here. To deal with this edges that cross feasibility boundaries are identified and the associated vertices which are outside of $\mathcal{X}$ are relocated to an appropriate point on the boundary of $\mathcal{X}$. Here any coordinate $i \in [n]$ of such an external vertex that is either less than l_i or greater than u_i is re-set to l_i or u_i as appropriate.

However the Voronoi approach has exponential dependence on n, as constructing the Voronoi diagram of $N_\mathcal{T}$ points requires $O(N_\mathcal{T} log N_\mathcal{T} + N_\mathcal{T}^{\lceil n/2 \rceil})$ time [Cha93]. This suggests that such an approach is not computationally viable for anything other than low dimensional problems. On the basis that a Voronoi diagram based approach is the primary recognised heuristic for identifying the largest empty *circle* we will consider a Voronoi based robust LEH heuristic here only in the context that for 2D problems in our experimental analysis this approach will serve as a good direct comparator for our other robust LEH heuristics.

2.4.3 Inner search methods

Discussions of the LEH approach have so far focussed on the outer minimisation search, assuming some form of inner search that provides the inner robust maximum for each candidate point in the minimisation search. In [BNT10b] a two-stage gradient ascent search is recommended for each inner search around a candidate point. This assumes gradient information is available and proposes $(n + 1)$ individual two-stage gradient ascents for each candidate. For a 100-dimensional problem this would require several thousand function evaluations around each candidate point. In practical terms both the number of function evaluations required to undertake a global search and the requirement for gradient information may make such extensive inner searches prohibitive. Given, for example, budgetary restrictions on the number of function evaluations, some trade-off must be achieved between the extent of each inner Γ-radius uncertainty neighbourhood search and globally exploring the search space. But this trade-off between robustness in terms of the extent of the inner searches, and performance in terms of the outer global search, is complex, see [MLM15, EDHX17]. For example the determination of an appropriate inner approach – type of search, extent of search and parameter settings –

may be both instance (problem and dimension) dependent and dependent on the outer approach.

Here we do not propose to recommend a definitive inner search approach. From a theoretical point of view we assume the information is provided by some oracle. From an experimental point of view in the algorithm testing and comparisons below we assume the same basic inner Γ-radius uncertainty neighbourhood analysis for all heuristics, to ensure a consistency when comparing results for alternative search approaches.

There is, however, an aspect of our LEH approach that enables an additional feature, the forcing of an early end to an inner search. The LEH approach is exploration-led, the objective being to locate and move to the candidate point in the decision variable space furthest from all hcps. Hcps are designated based on the determination of a high cost threshold τ, set here as the current estimate of the robust global minimum (min max) value. The nature of this approach enables (inner) uncertainty neighbourhood searches around each candidate point to be restricted when appropriate. If an inner search identifies a local point with objective function value above τ the inner search can be immediately curtailed on the basis that the candidate is not distant from hcps. This equates to the recognition that the candidate point is not an improvement on the current estimated robust optima. Such regulating of inner searches has the potential to significantly reduce the number of function evaluations expended on local neighbourhood analysis. In the case of budgetary limitations on numbers of function evaluations this further enables more exploration of the decision variable space.

2.4.4 Algorithm summary

Given one of our three approaches to identifying the LEH devoid of hcps, random, GA or Voronoi, the overarching algorithm for the robust exploratory LEH heuristic is given in Algorithm 1. Here one of these three approaches to the outer search is applied in line 16 as $LEH\_Calculator(H_\tau)$, for a defined high cost set H_τ. It is assumed that this routine will return a candidate point $\boldsymbol{x}_{LEH}$ and an associated radius r_{LEH}, that is the minimal distance between $\boldsymbol{x}_{LEH}$ and all points in H_τ. The heuristic will halt if r_{LEH} is not greater than Γ.

For a defined number of initialisation points, random points in $\mathcal{X}$ are selected and the function f evaluated at these points. The points and their function evaluations are recorded in history sets H and F_H, lines 1 - 6. Having randomly selected a candidate point $\boldsymbol{x}_c$ from H we perform an inner maximisation in the Γ-uncertainty neighbourhood around $\boldsymbol{x}_c$, see line 10. The description of the inner maximisation is given below as Algorithm 2. If this is the first candidate point, or the local robust value for this candidate $\tilde{g}(\boldsymbol{x}_c)$ is less than the current best solution τ, this minimum is updated and the associated global minimum point $\boldsymbol{x}_{Op}$ replaced by $\boldsymbol{x}_c$, see lines 11 - 14.

Next the high cost set H_τ is established as all members of H with corresponding function values in F_H that are greater than or equal to the current high cost threshold τ, see line 15. Based on H_τ, the next candidate point is identified via one of the outer search approaches, see line 16. If the heuristic is halted at this stage due to an inability to identify a valid LEH or at any stage due to the budget being exceeded, the extant estimate for the robust global minimum $\boldsymbol{x}_{Op}$ is returned.

Algorithm 1 Robust global exploration using Largest Empty Hyperspheres

```
    Input: $f$, $\mathcal{X}$, $\Gamma$
    Parameters: $Num\_Initial$, $Budget$, $Max\_Search$
 1: for all $i$ in $[Num\_Initial]$ do
 2:     Choose random point $\boldsymbol{x}^i \in \mathcal{X}$
 3:     Calculate $f(\boldsymbol{x}^i)$ and store in $F_H$
 4:     $Budget \leftarrow Budget - 1$
 5:     $H \leftarrow H \cup \{\boldsymbol{x}^i\}$
 6: end for
 7: Select random point $\boldsymbol{x}_c \in H$
 8: $r_{LEH} \leftarrow \infty$; $\tau \leftarrow \infty$
 9: while $r_{LEH} > \Gamma$ do
10:     $\tilde{g}(\boldsymbol{x}_c) \leftarrow$ CALL Algorithm 2
11:     if $\tilde{g}(\boldsymbol{x}_c) < \tau$ then
12:         $\boldsymbol{x}_{Op} \leftarrow \boldsymbol{x}_c$
13:         $\tau \leftarrow \tilde{g}(\boldsymbol{x}_c)$
14:     end if
15:     $H_\tau \leftarrow \{\boldsymbol{x} \in H : F_H(\boldsymbol{x}) \geq \tau\}$
16:     Find $(\boldsymbol{x}_{LEH}, r_{LEH})$ by calling LEH_Calculator($H_\tau$)
17:     $\boldsymbol{x}_c \leftarrow \boldsymbol{x}_{LEH}$
18: end while
19: return A robust solution $\boldsymbol{x}_{Op}$ and robust objective estimate $\tau$
```

Algorithm 2, the Γ-uncertainty neighbourhood inner maximisation called in line 10 of Algorithm 1, requires several inputs: *Budget* the current count of function evaluations completed, *Max_Search* the maximum number of function evaluations permitted in an inner search, $\boldsymbol{x}_c$ the current candidate point (centre of an LEH) around which the inner search is to be performed, Γ to define the uncertainty neighbourhood of $\boldsymbol{x}_c$, and τ the high cost threshold for stopping the inner search if appropriate.

Algorithm 2 proceeds by looping through up to *Max_Search* inner search points, identifying a point in the Γ-uncertainty neighbourhood of $\boldsymbol{x}_c$ and evaluating the function at each point visited, lines 10 - 22. Here the point to be evaluated is determined by random sampling in the Γ-radius hypersphere centred on $\boldsymbol{x}_c$, line 11. Under other inner maximisation rules this would be determined by some explicit maximisation search heuristic. As the function is evaluated at the inner search points the local robust value (inner maximum) *Local_Robust* is updated as appropriate, line 18. If *Local_Robust* exceeds the high cost threshold τ the inner maximisation is immediately terminated, lines 19 - 21. Algorithm 2 ends by returning an estimate for the worst case cost value at $\boldsymbol{x}_c$, $\tilde{g}(\boldsymbol{x}_c)$ into Algorithm 1.

Algorithm 2 Γ-uncertainty neighbourhood inner maximisation

```
    Input: $Budget$, $Max\_Search$, $\boldsymbol{x}_c$, $\Gamma$, $\tau$

 1: if $\tau < \infty$ then
 2:     Calculate $f(\boldsymbol{x}_c)$ and store in $F_H$
 3:     $H \leftarrow H \cup \{\boldsymbol{x}_c\}$
 4:     $Budget \leftarrow Budget - 1$
 5:     if $Budget == 0$ then
 6:         GOTO line 19 of Algorithm 1
 7:     end if
 8: end if
 9: Set $Local\_Robust \leftarrow f(\boldsymbol{x_c})$
10: for all $i$ in $[Max\_Search]$ do
11:     Choose $\Delta\boldsymbol{x}_c^i \in \mathcal{U}$, set $\boldsymbol{x}^i \leftarrow \boldsymbol{x}_c + \Delta\boldsymbol{x}_c^i$
12:     Calculate $f(\boldsymbol{x}_c^i)$ and store in $F_H$
13:     $H \leftarrow H \cup \{\boldsymbol{x}_c^i\}$
14:     $Budget \leftarrow Budget - 1$
15:     if $Budget == 0$ then
16:         GOTO line line 19 of Algorithm 1
17:     end if
18:     $Local\_Robust \leftarrow \max\{Local\_Robust, f(\boldsymbol{x}_c^i)\}$
19:     if $Local\_Robust > \tau$ then
20:         GOTO line 23
21:     end if
22: end for
23: $\tilde{g}(\boldsymbol{x}_c) \leftarrow Local\_Robust$
24: return $\tilde{g}(\boldsymbol{x}_c)$: estimated worst case cost at $\boldsymbol{x}_c$
```

2.4.5 Example LEH application

In order to give some indication of the nature of our LEH search we have applied it to the 2-dimensional problem (poly2D) and plotted the points evaluated and associated search path of the current estimate of the robust global minimum in Figures 2.4e and 2.4f. Here the LEH Voronoi algorithm is used. For comparison we have also plotted corresponding results for two alternative heuristics, a robust Particle Swarm Optimisation (PSO) approach shown in Figures 2.4a and 2.4b, and the local descent directions approach from Section 2.3.2 shown in Figures 2.4c and 2.4d. Here the robust PSO is used as a proxy to a brute force or co-evolutionary approach. The basic global PSO formulations have been used, as described in [SE98]. The descent directions approach has been extended by using random re-starts, as a proxy to extending it to a global approach. In all cases inner random sampling in a hypersphere of 100 Γ-uncertainty neighbourhood points is used, and a maximum budget of 10,000 function evaluations employed.

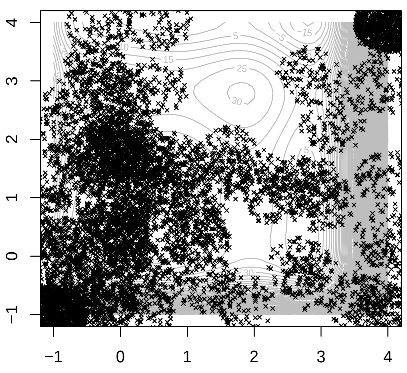
The plots shown in Figure 2.4 are for only a single run of each heuristic, and as such should only be seen as exemplars intended to give some indication of the different natures of these outer search approaches. It can be seen that whilst the robust PSO explores the decision variable space somewhat, and the re-starting descent directions follows (exploits) a series of local paths, the LEH approach features both considerable exploration globally and more intense analysis of promising points. It is clear that the curtailing of the inner searches in the LEH approach enables much wider exploration for fewer function evaluations. In this example less than 1,000 function evaluations have been required before the LEH heuristic has stopped because an LEH of radius greater than Γ cannot be found, but for larger (dimensional) problems such stopping prior to reaching the budgetary limit will not apply. One striking feature of Figure 2.4e is how many of the inner searches stop immediately on the evaluation of a candidate point. This is because the objective value at these candidate points exceeds the current threshold τ.

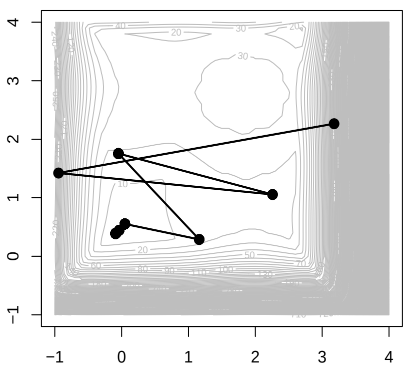
The Voronoi based search exemplified by Figures 2.4e and 2.4f is a good indicator of the nature of the searches due to all three LEH approaches, random, GA and Voronoi. However the radii of the LEH identified for each candidate will vary with the use of each of these algorithms. Figure 2.9 in Appendix 2.7.2 gives some indication of how the radii of the hyperspheres generated by each of these LEH heuristics progress as the exploration proceeds.

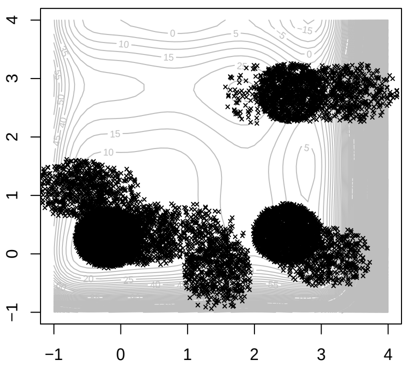
2.5 Computational experiments

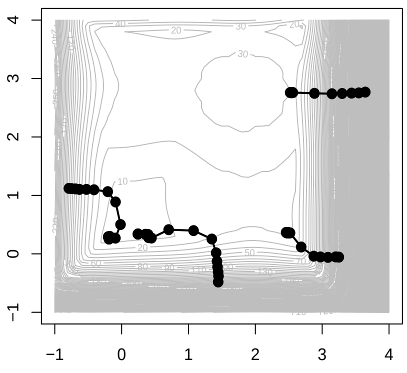
2.5.1 Set up

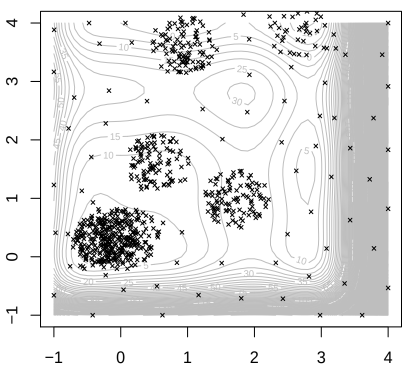
In order to assess the effectiveness of the LEH approach the heuristic has been applied to eight test problems, and results compared against the two alternative search heuristics described in Section 2.5.2. Experiments have been performed on 2D, 4D, 7D, 10D and 100D instances of these test problems; results have also been generated for (poly2D). Both the genetic algorithm and random forms of the LEH heuristic have been assessed for all instances. The LEH Voronoi has additionally been applied to the 2D instances, with the intention of giving some indication of the differences due to a 'best' LEH identifier algorithm (Voronoi) versus the alternatives. All LEH approaches are initialised by randomly sampling a single point in $\mathcal{X}$. Assuming that for most real-world problems


(a) PSO points


(b) PSO search


(c) DD points


(d) DD search


(e) LEH Vor points

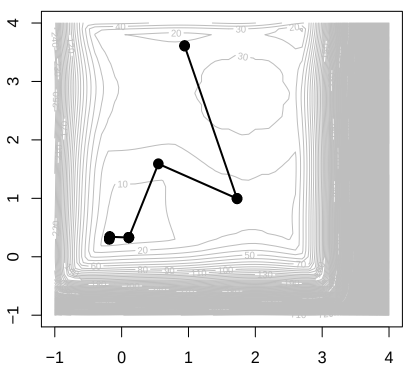

(f) LEH Vor search

Figure 2.4: Contour plots of example searches of the 2-dimensional problem (poly2D), for Γ=0.5. Plots on the left show all points evaluated. Plots on the right show the progress of the current best robust solution. The heuristics used are: (top) outer PSO, (middle) outer descent directions with re-start, and (bottom) outer LEH using the Voronoi based approach.

the optimisation analysis will be limited by resources, a fixed budget of 10,000 function evaluations (model runs) is assumed. The same inner approach is employed for all heuristics. A simple random sampling in a hypersphere of 100 points in a point's local Γ-uncertainty neighbourhood is used for all instances, and the local robust maximum is estimated as the maximum due to this sampling. For the LEH approaches this inner sampling is curtailed if a point is identified in the uncertainty neighbourhood that has objective value exceeding the current high cost threshold τ.

All experiments have have been performed using Java, on an HP Pavilion 15 Notebook laptop computer, with 64 bit operating system, an Intel Core i3-5010U, 2.10GHz processor, and 8GB RAM. Each heuristic search has been applied to each test problem-dimension instance 50 times to reduce variability. For the solution of the Second Order Cone Problem as part of the descent directions algorithm [BNT10b], the IBM ILOG CPLEX Optimization Studio V12.6.3 package is called from Java.

2.5.2 Comparator heuristics

Our experiments have been conducted on LEH, a re-starting descent directions, and robust PSO metaheuristics. We have applied parameter tuning to 3 of the 5 comparator heuristics – LEH Voronoi and LEH Random do no have tunable parameters – employing an evolutionary tuning approach using a genetic algorithm to generate a single set of parameters for each heuristic, for all test problems. For each of the 3 tuned heuristics the same subset of the test instances was used, running each member of an evolving population on each of these instances multiple times to generate mean result for each member of a population on each test instance. The performance of each individual in a population was ranked separately for each test instance, across the members of the population, leading to mean overall ranks which were used as the utility measure in tournament selection; see e.g. [ES12]. Tuned parameter values are given in Appendix 2.7.4.

The effectiveness of the local descent directions approach [BNT10b] suggests that extending this to a global search by using random re-starts will provide a reasonable comparator. A local descent directions search is undertaken from a random start point, and when this is complete it is repeated from another random start point. This is repeated until the function evaluations budget is reached. In descent directions a set of high cost points leads to the identification of an optimal stepping direction and step size, if a valid direction exists. However the algorithm includes a number of dynamically changing parameters which adapt the high cost set and enforce a minimum step size. Here we have tuned 5 parameters relating to these stages of the heuristic; see [BNT10b] for further information. Labelled 'd.d. Re' in the results section.

As a proxy to a brute force or co-evolutionary approach an outer particle swarm search is considered. The basic formulations for the global PSO approach have been used as described in [SE98] and 5 parameters have been tuned: swarm size, number of iterations, and for the velocity equation the C_1 and C_2 acceleration parameters and inertia weight parameter ω. The combined swarm size times number of iterations was limited to 100 in order to align with the budget of 10,000 function evaluations and the level of inner sampling. Labelled 'PSO' in the results section.

Our robust LEH metaheuristic is considered for the three alternative ways of identifying the largest hypersphere that is empty of hcps:

- Randomly sampled valid empty hypersphere, see Section 2.4.2.1. This includes re-sampling up to 1,000 potential candidates in an attempt to identify a valid empty hypersphere, otherwise it is assumed that a valid point cannot be found and a robust global minimum has been reached. Labelled 'LEH Rnd' in the results section.

- Genetic algorithm LEH, see Section 2.4.2.2. Here we have tuned 6 parameters: the size of the population, number of generations, number of elites, tournament size, and mutation probability and size; we have fixed the use of tournament selection and the choice of mid-point crossover. The combined population size times number of generations was limited to 100, which is somewhat based on run time considerations associated with the large value of N_τ, the number of candidate points visited with a budget of 10,000 function evaluations. Labelled 'LEH GA' in the results section.

- Voronoi based [Tou83] LEH, see Section 2.4.2.3. Here the construction of the Voronoi diagram for the input points H_τ is performed using the Java library due to [Nah17]. This generates geometric data, Voronoi vertices and edges, which are used to determine a set of potential candidate points – Voronoi vertices, including those originally outside of $\mathcal{X}$ relocated to the boundary of $\mathcal{X}$ – for the centre of the LEH. Labelled 'LEH Vor' in the results section.

2.5.3 Test functions

A large number of test functions are available for benchmarking optimisation algorithms, and posing a variety of difficulties, see [Kru12, JY13]. Here eight are considered, plus (poly2D) as outlined in Section 2.3.1. In each case a single Γ-uncertainty value is used:

- Ackleys: feasible region [-32.768, 32.768]; Γ=3.0.

- Multipeak F1: feasible region [0, 1]; Γ=0.0625.

- Multipeak F2: feasible region [0, 10]; Γ=0.5.

- Rastrigin: feasible region [-5.12, 5.12]; Γ=0.5.

- Rosenbrock: feasible region [-2.048, 2.048]; Γ=0.25.

- Sawtooth: feasible region [-1, 1]; Γ=0.2.

- Sphere: feasible region [-5, 5]; Γ=1.0.
- Volcano: feasible region [-10, 10]; Γ=1.5.

The full description of these eight test functions is given in Appendix 2.7.1. To give some indication of the nature of these functions contour plots of the 2D instances are shown in Figure 2.5, for both the nominal and worst cases.

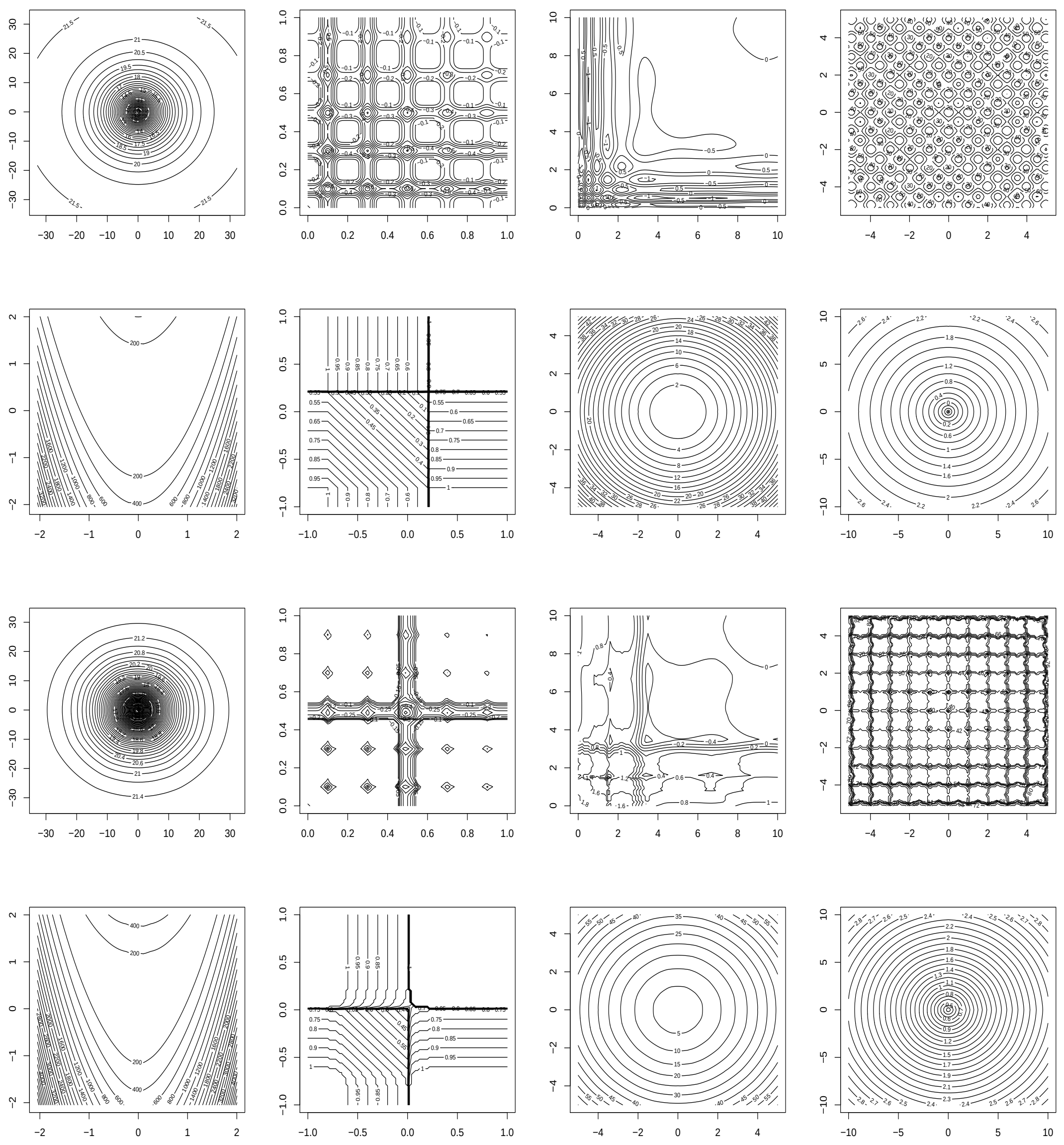


Figure 2.5: Contour plots of nominal (top 8) and worst case (bottom 8) 2D test functions. Left to right, top to bottom: Ackley, Multipeak F1, Multipeak F2, Rastrigin, Sawtooth, Sphere and Volcano.

2.5.4 Results

Results of the 50 sample runs for each heuristic applied to each test problem-dimension instance are presented here. In each run the best solution as identified by the heuristic is used. However the points in the decision variable space that have been identified as best have robust values generated using the simple inner random sampling approach, with a budget of up to 100 sample points. To better approximate the true robust values at these points their robust values have been re-estimated based on randomly sampling a large number of points (nominally 1,000,000) in the Γ-uncertainty neighbourhood of the identified robust point. This is a post processing exercise and does not affect the min max search.

Mean results due to each set of 50 sample runs are shown in Tables 2.1 and 2.2. We have applied the Wilcoxon rank-sum test with 95% confidence to identify the statistically best approaches. Results highlighted in bold indicate the approaches that are statistically equivalent to the best one observed, for a given problem-dimension instance. Corresponding box plots, giving some indication of how the results are distributed across the 50 samples, are shown in Figures 2.6, 2.7 and 2.8. Additional results, the standard deviations due to each set of 50 sample runs, the average number of candidate points visited and average number of function evaluations undertaken, are given in Appendix 2.7.3.

		(poly2D)
	PSO	5.57
	d.d. Re	**5.11**
2D	LEH Vor	**5.52**
	LEH GA	5.50
	LEH Rnd	**5.26**

Table 2.1: Mean results due to 50 sample runs for the 2-dimensional polynomial function (poly2D) due to [BNT10b].

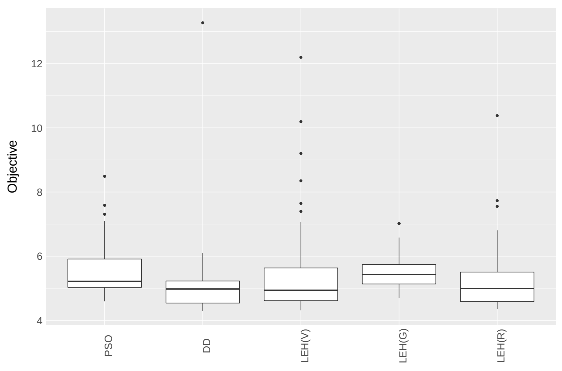


Figure 2.6: Box plots of robust objective values due to multiple sample runs for the 2-dimensional polynomial function (poly2D) due to [BNT10b].

		Ackley's	MultipeakF1	MultipeakF2	Rastrigin	Rosenbrock	Sawtooth	Sphere	Volcano
2D	PSO	11.44	-0.36	-0.49	38.04	10.00	**0.49**	1.47	0.39
	d.d. Re	12.78	-0.40	-0.44	36.42	**7.71**	0.54	**1.01**	**0.24**
	LEH Vor	**9.36**	**-0.61**	**-0.68**	**34.67**	**7.71**	0.59	1.05	**0.24**
	LEH GA	9.62	-0.60	-0.65	**35.17**	**7.68**	**0.48**	1.14	0.27
	LEH Rnd	9.77	-0.59	-0.65	35.52	7.92	**0.47**	1.21	0.29
4D	PSO	13.50	-0.30	-0.36	65.91	34.20	0.50	3.35	0.75
	d.d. Re	17.32	-0.33	-0.32	60.43	**11.94**	0.60	**1.02**	**0.46**
	LEH GA	**8.73**	**-0.64**	**-0.68**	**54.34**	12.17	**0.45**	1.39	**0.34**
	LEH Rnd	12.21	-0.50	-0.57	61.39	23.18	0.46	1.70	0.57
7D	PSO	15.36	-0.29	-0.23	102.35	123.42	0.51	8.21	1.27
	d.d. Re	19.72	-0.30	-0.24	**88.44**	**17.47**	0.63	**1.03**	1.21
	LEH GA	**12.35**	**-0.51**	**-0.57**	**88.07**	48.75	**0.42**	2.94	**0.77**
	LEH Rnd	16.19	-0.42	-0.48	104.31	126.28	0.52	9.49	1.37
10D	PSO	16.17	-0.31	-0.15	142.99	238.36	0.51	14.66	1.63
	d.d. Re	20.69	-0.30	-0.19	**112.61**	**41.12**	0.63	**1.40**	1.93
	LEH GA	**14.08**	**-0.48**	**-0.56**	**115.06**	103.31	**0.43**	7.34	**1.19**
	LEH Rnd	18.11	-0.39	-0.43	145.52	322.27	0.55	20.62	1.92
100D	PSO	19.02	-0.35	-0.17	1,215.34	7,989.77	0.49	226.66	4.45
	d.d. Re	21.38	-0.32	-0.32	1,386.77	36,141.80	0.70	656.86	6.18
	LEH GA	**17.30**	**-0.44**	**-0.42**	**1,065.44**	**3,264.49**	**0.43**	**136.18**	**3.79**
	LEH Rnd	21.12	-0.36	-0.28	1,577.84	26,526.42	0.66	588.03	5.93

Table 2.2: Mean results due to 50 sample runs.

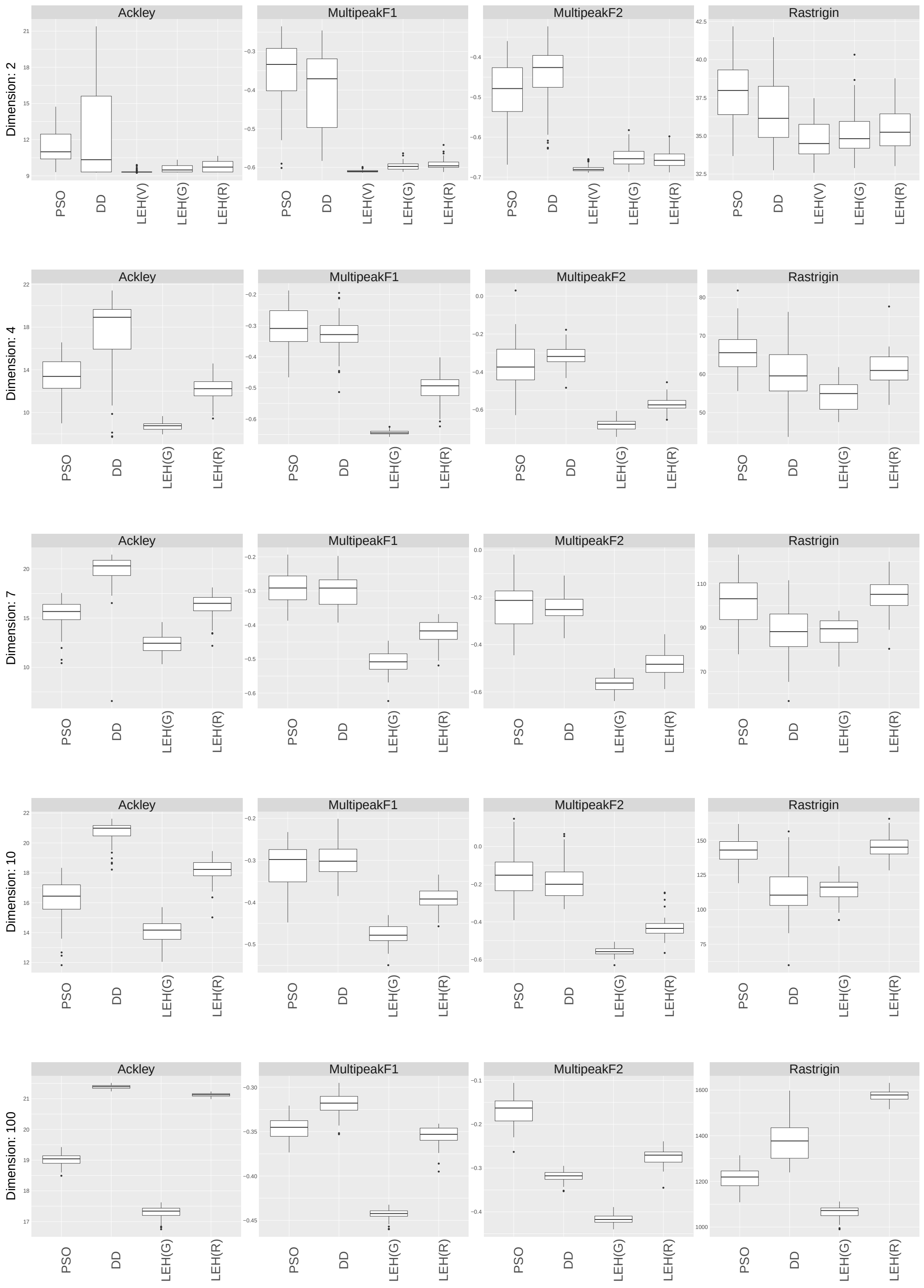


Figure 2.7: Box plots of robust objective values due to multiple sample runs. Left to right: Ackleys, Multipeak F1, Multipeak F2, Rastrigin; Top to bottom: 2D, 4D, 7D, 10D, 100D.

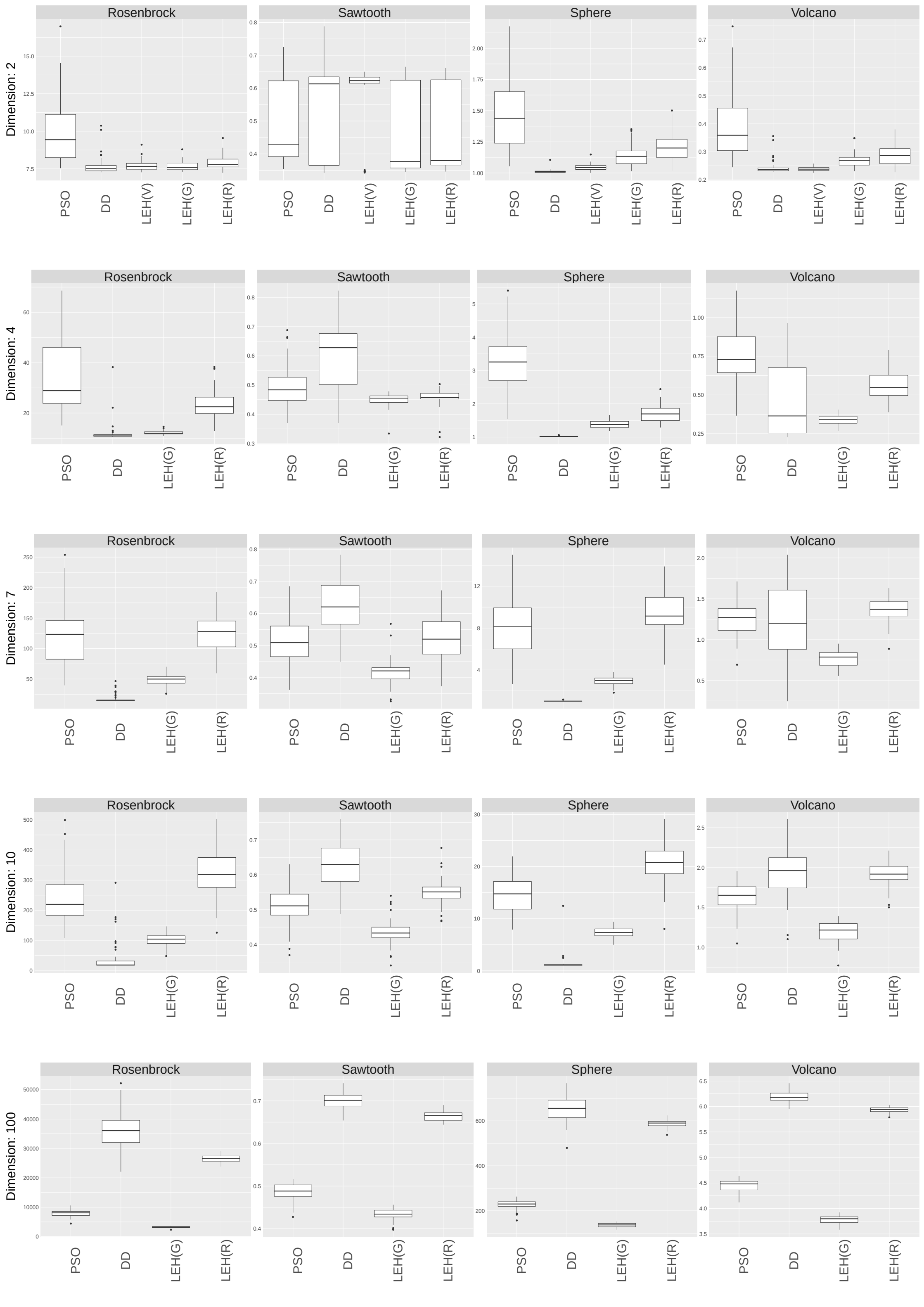


Figure 2.8: Box plots of robust objective values due to multiple sample runs. Left to right: Rosenbrock, Sawtooth, Sphere, Volcano; Top to bottom: 2D, 4D, 7D, 10D, 100D.

From Table 2.2 we see that for 100D instances the LEH GA approach is best for all test problems, and in several cases the mean value for LEH GA is substantially lower than for all of the alternative heuristics. From Tables 2.1 and 2.2 the LEH approach is among the best in at least 6 of the instances for all other dimensions.

For 2D instances the LEH Voronoi approach is among the best results for 7 of the 9 problems, whilst LEH GA and LEH Rnd are each amongst the best results for 3 and 2 problems respectively. It should also be noted that in 5 of the 7 instances where LEH Voronoi is among the best, LEH GA is either statistically equivalent or the mean value is second best. For the 2D Sphere instance d.d. Re is best with LEH Voronoi second best, whilst d.d. Re and LEH heuristics are statistically equivalent for the (poly2D) and 2D Volcano and Rosenbrock instances. The robust PSO approach is statistically equivalent to LEH heuristics for the 2D Sawtooth instance.

For the 4D – 10D instances d.d. Re is statistically equivalent to LEH GA in the 4D Volcano problem and the 7D and 10D instances of the Rastrigin problem. For the 4D – 10D Rosenbrock and Sphere instances d.d. Re is best and LEH GA second best, with the mean value for d.d. Re substantially lower in the 7D and 10D cases. Considering the shape of the Rosenbrock and Sphere functions it can be expected that a local search will perform particularly well for these problems.

LEH GA is better than LEH Rnd for all instances excluding (poly2D). In a number of instances the mean value for LEH GA is substantially lower than the mean value for LEH Rnd. The number of candidate points that LEH can visit is substantially increased by the early stopping of inner searches as soon as the high cost threshold is exceeded, see Tables 2.3 and 2.5 in Appendix 2.7.3. Although this feature must unquestionably play a role in the success of the LEH GA approach, the fact that LEH Rnd visits a comparable number of candidate points indicates that the additional pro active seeking of the largest hypersphere devoid of high cost points is also a significant factor in the success of LEH GA.

2.6 Conclusions and further work

We have introduced a new metaheuristic for box-constrained robust optimisation problems with implementation uncertainty. We do not assume any knowledge on the structure of the original objective function, making the approach applicable to black-box and simulation-optimisation problems. We do assume that the solution is affected by uncertainty, and the aim is to find a solution that optimises the worst possible performance in this setting. This is the min max problem. Previously, few generic search methods have been developed for this setting.

We introduce a new approach for a global search based on distinguishing undesirable high cost – high objective value – points (hcps), identifying the largest hypersphere in the decision variable space that is completely devoid of hcps, and exploring the decision variable space by stepping between the centres of these largest empty hyperspheres.

We demonstrated the effectiveness of the approach using a series of test problems, considering instances of varying dimension, and comparing our LEH approach against one metaheuristic that employs an outer particle swarm optimisation and one from the literature that uses multiple re-starts of the local descent directions approach. For low and moderate dimensional instances the approach shows competitive performance; for high-dimensional problems the LEH approach significantly outperforms the comparator heuristics for all problems.

There are several ways in which this work can be developed. Further consideration can be given to the inner maximisation search approach in order to better understand the trade-off between expending function evaluations on the local Γ-radius uncertainty neighbourhood search versus globally exploring the search space, in the context of our LEH approach.

The repeated calculation of large numbers of Euclidean distances each time a new LEH needs to be identified within the LEH GA heuristic is computationally expensive. Rather than only calculating a single next candidate point each time the GA is performed, identifying multiple points could speed up computation or alternatively enable the use of larger population-generation sizes to improve the estimation of the largest empty hypersphere.

Results of the mid-dimension experiments on the Rosenbrock and Sphere test problems suggest that an exploitation based approach works well in these instances, indicating a direction for extending our exploration focussed LEH approach.

It is clear that within the LEH algorithm the early stopping of the inner searches when it is established that the current robust global value cannot be improved upon has significant advantages. It is worth considering whether alternative search approaches could take advantage of this feature.

In addition to the test problems considered here, as further research it would be beneficial to apply our LEH GA approach to a real-world problem.

2.7 Appendices

2.7.1 Test functions

Functions used to assess the effectiveness of the Largest Empty Hypersphere robust metaheuristics taken from [Kru12, JY13].

Ackleys

$$f(\boldsymbol{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + \exp(1)$$

The feasible region is the hypercube $x_i \in$ [-32.768, 32.768].

MultipeakF1

$$f(\boldsymbol{x}) = -\frac{1}{n}\sum_{i=1}^{n} g(x_i)\ ,\ g(x_i) = \begin{cases} \exp(2\ln 2(\frac{x_i-0.1}{0.8})^2)\sqrt{|\sin(5\pi x_i)|} & \text{if } 0.4 < x_i \leq 0.6\ , \\ \exp(2\ln 2(\frac{x_i-0.1}{0.8})^2)\sin^6(5\pi x_i) & \text{otherwise} \end{cases}$$

The feasible region is the hypercube $x_i \in$ [0, 1].

MultipeakF2

$$f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} g(x_i)\ ,\ g(x_i) = 2\sin(10\exp(-0.2x_i)x_i)\exp(-0.25x_i)$$

The feasible region is the hypercube $x_i \in$ [0, 10].

Rastrigin

$$f(\boldsymbol{x}) = 10n + \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)\right]$$

The feasible region is the hypercube $x_i \in$ [-5.12, 5.12].

Rosenbrock

$$f(\boldsymbol{x}) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

The feasible region is the hypercube $x_i \in$ [-2.048, 2.048].

Sawtooth

$$f(\boldsymbol{x}) = 1 - \frac{1}{n}\sum_{i=1}^{n} g(x_i)\ ,\ g(x_i) = \begin{cases} x_i + 0.8 & \text{if } -0.8 \leq x_i < 0.2\ , \\ 0 & \text{otherwise} \end{cases}$$

The feasible region is the hypercube $x_i \in$ [-1, 1].

Sphere

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2$$

The feasible region is the hypercube $x_i \in$ [-5, 5].

Volcano

$$f(\boldsymbol{x}) = \begin{cases} \sqrt{\|\boldsymbol{x}\|} - 1 & \text{if } \|\boldsymbol{x}\| > 1 \; , \\ 0 & \text{otherwise} \end{cases}$$

The feasible region is the hypercube $x_i \in$ [-10, 10].

2.7.2 Radii due to alternative LEH algorithms

Whilst the Voronoi based search exemplified by Figures 2.4e and 2.4f in Section 2.4.5 is a good indicator of the nature of the searches due to all three alternative LEH approaches, random, GA and Voronoi, the radii of the LEH identified for each candidate will vary across these approaches. Here Figure 2.9 gives some indication of how the radii of the hyperspheres generated by each these three LEH heuristics progress as the exploration proceeds. The three curves represent separate runs of the LEH algorithm when applied to (poly2D),and should be considered indicative.

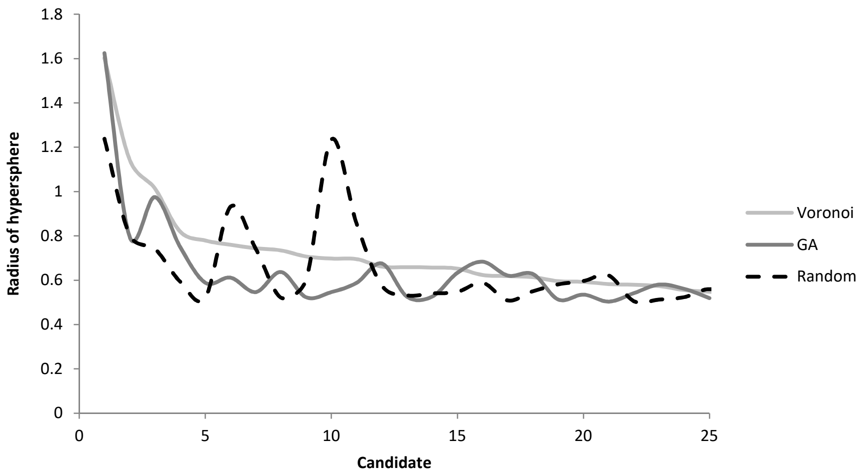


Figure 2.9: Alternative LEH approaches applied to the same problem: variation in empty hypersphere radius with numbers of candidates evaluated for robustness.

As would be expected the general nature of the size of the radius of the LEH steadily decreases with increasing numbers of candidate points evaluated. However superimposed on this overall decrease are the indicative patterns due to the alternative heuristics. For the random algorithm the size of the LEH is quite variable, whilst for Voronoi the curve is smooth. The GA algorithm sits somewhere between the two.

2.7.3 Additional results

The standard deviations due to each set of 50 sample runs, the average number of candidate points visited and average number of function evaluations undertaken, are shown in Tables 2.3, 2.4, 2.5 and 2.6 here respectively. Labelling of comparator heuristics in the tables is as follows:

- PSO: particle swarm optimisation.
- d.d. Re: Multi re-start descent directions.
- LEH Vor: LEH using a Voronoi [Tou83] approach; applied to 2D problems only.
- LEH GA: LEH using a genetic algorithm.
- LEH Rnd: LEH using random sampling.

		Std. dev.	Candidates	Evaluations
	PSO	0.83	100	10,000
	d.d. Re	1.27	100	10,000
poly2D	LEH Vor	1.60	35	995
	LEH GA	0.56	30	727
	LEH Rnd	1.07	35	1,037

Table 2.3: Standard deviations of results, average number of candidate points visited, and average number of points evaluated for the 50 sample runs for the 2-dimensional polynomial function (poly2D) due to [BNT10b].

		Ackley's	MultipeakF1	MultipeakF2	Rastrigin	Rosenbrock	Sawtooth	Sphere	Volcano
2D	PSO	1.35	0.09	0.08	2.04	2.17	0.12	0.30	0.11
	d.d. Re	4.14	0.10	0.08	2.26	0.62	0.14	0.02	0.02
	LEH Vor	0.15	0.00	0.01	1.23	0.34	0.10	0.03	0.01
	LEH GA	0.35	0.01	0.02	1.53	0.31	0.14	0.09	0.03
	LEH Rnd	0.45	0.01	0.02	1.43	0.47	0.13	0.11	0.04
4D	PSO	1.66	0.06	0.13	5.79	14.21	0.08	0.89	0.18
	d.d. Re	3.55	0.06	0.06	7.29	4.17	0.11	0.01	0.25
	LEH GA	0.38	0.01	0.03	4.04	0.84	0.02	0.13	0.03
	LEH Rnd	1.19	0.05	0.04	4.47	5.30	0.03	0.26	0.08
7D	PSO	1.54	0.05	0.10	10.70	49.51	0.06	2.66	0.21
	d.d. Re	2.21	0.05	0.07	11.08	7.19	0.08	0.03	0.47
	LEH GA	1.06	0.03	0.03	6.58	9.90	0.04	0.44	0.10
	LEH Rnd	1.26	0.04	0.05	8.20	31.15	0.07	2.05	0.13
10D	PSO	1.43	0.05	0.13	10.06	89.74	0.06	3.36	0.19
	d.d. Re	0.83	0.04	0.09	16.48	53.56	0.06	1.63	0.29
	LEH GA	0.82	0.03	0.02	7.95	20.73	0.04	1.08	0.13
	LEH Rnd	0.79	0.03	0.06	7.93	80.92	0.04	3.77	0.14
100D	PSO	0.20	0.01	0.03	44.69	1284.24	0.02	20.84	0.13
	d.d. Re	0.06	0.01	0.01	97.02	6518.29	0.02	59.59	0.11
	LEH GA	0.20	0.01	0.01	27.87	251.68	0.01	9.15	0.08
	LEH Rnd	0.06	0.01	0.02	26.49	1289.96	0.01	16.83	0.06

Table 2.4: Standard deviations of results due to 50 sample runs.

		Ackley's	MultipeakF1	MultipeakF2	Rastrigin	Rosenbrock	Sawtooth	Sphere	Volcano
	PSO	100	100	100	100	100	100	100	100
	d.d. Re	100	100	100	100	100	100	100	100
2D	LEH Vor	245	107	205	309	129	26	24	63
	LEH GA	147	78	117	160	90	36	33	57
	LEH Rnd	213	98	183	256	109	26	23	53
	PSO	100	100	100	100	100	100	100	100
4D	d.d. Re	100	100	100	100	100	100	100	100
	LEH GA	9,054	7,543	8,355	3,624	8,433	1,304	1,334	3,890
	LEH Rnd	8,799	7,072	7,770	5,358	8,644	6,264	6,408	8,646
	PSO	100	100	100	100	100	100	100	100
7D	d.d. Re	100	100	100	100	100	100	100	100
	LEH GA	9,097	7,958	8,754	5,522	9,062	7,772	9,164	9,184
	LEH Rnd	8,799	7,341	7,819	6,821	8,781	8,229	8,789	8,816
	PSO	100	100	100	100	100	100	100	100
10D	d.d. Re	100	100	100	100	100	100	100	100
	LEH GA	9,063	7,822	8,716	6,526	9,039	7,977	9,037	9,086
	LEH Rnd	8,746	7,674	7,679	7,367	8,874	8,279	8,734	8,773
	PSO	100	100	100	100	100	100	100	100
100D	d.d. Re	100	100	100	100	100	100	100	100
	LEH GA	8,902	8,779	8,765	8,615	8,848	8,551	8,907	8,877
	LEH Rnd	7,805	8,679	8,533	8,708	8,954	8,708	8,889	8,827

Table 2.5: Average number of candidate points visited across 50 sample runs.

		Ackley's	MultipeakF1	MultipeakF2	Rastrigin	Rosenbrock	Sawtooth	Sphere	Volcano
2D	PSO	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000
	d.d.Re	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000
	LEH Vor	945	996	1,164	2,655	1,102	807	546	577
	LEH GA	830	890	958	1,835	871	780	686	696
	LEH Rnd	1,255	1,250	1,306	2,521	1,098	1,110	934	1,021
4D	PSO	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000
	d.d.Re	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000
	LEH GA	10,000	9,198	10,000	10,000	9,822	2,899	2,279	4,866
	LEH Rnd	10,000	10,000	10,000	10,000	10,000	8,477	8,031	10,000

Table 2.6: Average number of points evaluated in the 50 sample runs. For all test problems of dimension 7 or higher the full budget of 10,000 function evaluations was used in all sample runs.

2.7.4 Heuristic parameter values

As described in Section 2.5.2 the parameter values for 3 of the heuristics – d.d. Re, PSO and LEH GA – have been tuned in order to generate a single set of parameters for each, which were then used in the generation of the experimental results given in Section 2.5.4. The values of those tuned parameters are given below in Tables 2.7 to 2.9.

For the d.d. Re heuristic 5 parameters were tuned: σ_{init} the initialisation factor for the high cost set threshold $\sigma(t)$, the $\sigma(t)$ reduction factor α used to adapt the threshold value, σ_α a lower threshold for $\sigma(t)$, a factor $init\rho_{Min}$ which is multiplied by the uncertainty parameter Γ in order to establish a minimum step size for the search, and ρ_{red} a reduction factor for reducing this minimum step size with every step t. See [BNT10b] for a full description of these parameters. The tuned parameter values are given in Table 2.7.

	σ_α	α	σ_{init}	ρ_{red}	$init\rho_{Min}$
d.d. Re	0.0065	1.059	0.1979	0.9456	0.0396

Table 2.7: Tuned parameter values for the d.d. Re heuristic.

For the PSO heuristic 5 parameters were also tuned: the C_1 and C_2 acceleration parameters, the inertia weight parameter ω, and the swarm size and number of iterations over which the swarm moves. See [SE98] for a further description of these parameters. The tuned parameter values are given in Table 2.8.

	C_1	C_2	ω	swarm	iterations
PSO	1.845	0.975	0.189	20	5

Table 2.8: Tuned parameter values for the PSO heuristic.

For the LEH GA heuristic which employs a genetic algorithm to search for the centre of the largest empty hypersphere, 6 parameters associated with the GA were tuned: the size of the population, number of generations, number of elites, tournament size, and mutation probability and size. Here mutation 'size' is actually a percentage value which is subsequently multiplied by the dimensional range of the decision variable space $\mathcal{X}$ in order to specify the actual amount by which any value is adjusted due to mutation. The tuned parameter values are given Table 2.9.

	population	generations	elites	tour size	mut prob	mut size
LEH GA	20	5	0	3	0.2624	0.175

Table 2.9: Tuned parameter values for the LEH GA heuristic.

Chapter 3

Paper 2: Particle Swarm Metaheuristics for Robust Optimisation with Implementation Uncertainty

Accepted for publication in the journal *Computers & Operations Research* in May 2020 [HGD20b].
Partially funded through EPSRC grants EP/L504804/1 and EP/M506369/1.

Author 1: Martin Hughes, Lancaster University, United Kingdom.
Author 2: Marc Goerigk, University of Siegen, Germany.
Author 3: Trivikram Dokka, Lancaster University, United Kingdom.

Abstract: We consider global non-convex optimisation problems under uncertainty. In this setting, it is not possible to implement a desired solution exactly. Instead, any other solution within some distance to the intended solution may be implemented. The aim is to find a robust solution, i.e., one where the worst possible solution nearby still performs as well as possible.

Problems of this type exhibit another maximisation layer to find the worst case solution within the minimisation level of finding a robust solution, which makes them harder to solve than classic global optimisation problems. So far, only few methods have been provided that can be applied to black-box problems with implementation uncertainty. We improve upon existing techniques by introducing a novel particle swarm based framework which adapts elements of previous methods, combining them with new features in order to generate a more effective approach. In computational experiments, we find that our new method outperforms state of the art comparator heuristics in almost 80% of cases.

3.1 Introduction

Decision making in the face of uncertainty is a widespread challenge. In many real-world situations it is common practice to use models to support informed decision making. However model run times and the extent of the decision variable solution space may render an extensive assessment of the problem space computationally impractical. In such circumstances an efficient global optimisation search approach is needed. The consideration of uncertainty in the modelling process, reflecting uncertainty in the real-world problem, may impact model outputs and therefore the optimum objective function value. Thus uncertainty adds an additional feature into any global optimisation search. Whilst simply ignoring the uncertainty is one strategy, such an approach has been shown to produce sub-optimal results, see [BTEGN09, GS16]. An approach is required that can identify a solution that performs well over a range of scenarios as opposed to simply in the expected case.

The use of a model in the form of a mathematical program is preferable from an optimisation standpoint, as such models may be solved efficiently with the determination of global optima guaranteed. However such an approach necessitates that the problem at hand can be adequately expressed in the form of a mathematical program. In many real-world situations this is not possible. Rather some more general form of simulation model will be used, which from an optimisation perspective, may be considered a black-box: decision variables values are input and an objective value is output. For such black-box problems a more general search technique that can be applied to any model is required, such as a metaheuristic. Such an approach accommodates a complete lack of knowledge of the structure of the model and of the nature of the corresponding objective function surface.

Optimisation under uncertainty is typically approached using either stochastic or robust techniques. In stochastic optimisation the probability distributions of the uncertain parameters are assumed to be known and the fitness of any solution is determined by some statistical measure, see [PBJ06, HdMB14]. By contrast robust optimisation only assumes that all uncertainty scenarios can be described by some set [BTN98]. A classic robust approach finds a solution that optimises its performance in the worst case. This is known as min max: at any point in the decision variable space an inner objective is employed to identify the maximal function value in the point's uncertainty neighbourhood, with an outer objective employed to identify the minimum maximal value.

Here we develop a new metaheuristic framework for the robust global optimisation of black-box problems, including non-convex problems. We assume no knowledge of the structure of the underlying model. The algorithm accommodates implementation uncertainty, where a desired solution may be somewhat perturbed in a real-world setting. We adopt the classic robust worst case approach. More formally, the general optimisation problem to be considered here is:

$$\begin{aligned} &\min\ f(\boldsymbol{x}) \\ &\text{s.t. } \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ is the n-dimensional vector of decision variables, and $\mathcal{X} \subseteq \mathbb{R}^n$ is the set of feasible solutions. We use the notation $[n] := \{1, \ldots, n\}$ and assume box constraints $\mathcal{X} = \prod_{i \in [n]} [l_i, u_i]$. A penalty in the objective is assumed in the case of other feasibility constraints. Such a problem, without any consideration of uncertainty, is designated the nominal problem here.

As an example, consider a non-convex one-dimensional problem due to [Kru12]. For the nominal problem, shown in Figure 3.1a, some standard metaheuristic could be used to locate the global minimum at $\boldsymbol{x}_0$. However if the solution that a decision maker wants to implement is somewhat perturbed in practice, the potential impact on the identification of the global minimum needs to be taken into consideration. The sensitivity of the objective to variations in the region of $\boldsymbol{x}_0$ is of particular concern, as highlighted in Figure 3.1b.

We assume only a solution similar to the desired solution $\boldsymbol{x}$, $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \Delta\boldsymbol{x}$ may be achieved. The classic robust approach is then to find a robust solution $\boldsymbol{x}$ such that for any $\tilde{\boldsymbol{x}}$ from the uncertainty neighbourhood of $\boldsymbol{x}$, the worst performance is optimised. As common in the literature (see, e.g., [BNT10b]), we consider the following uncertainty neighbourhood (also known as uncertainty set):

$$\mathcal{U} := \{\Delta\boldsymbol{x} \in \mathbb{R}^n \mid \|\Delta\boldsymbol{x}\| \leq \Gamma\}$$

where $\Gamma > 0$ defines the radius of the uncertainty neighbourhood around a solution $\boldsymbol{x} \in \mathcal{X}$ and $\|\cdot\|$ represents the Euclidean norm. The worst case costs of a solution $\boldsymbol{x}$ are:

$$g(\boldsymbol{x}) := \max_{\Delta\boldsymbol{x} \in \mathcal{U}} f(\boldsymbol{x} + \Delta\boldsymbol{x})$$

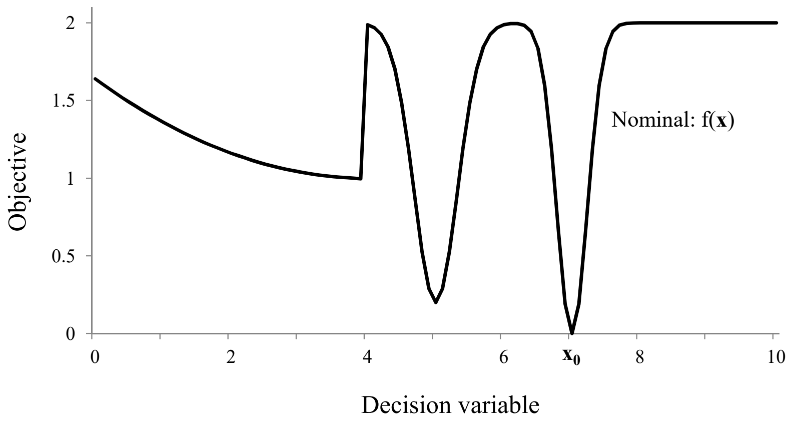
The min max robust optimisation problem is then:

$$\min_{\boldsymbol{x} \in \mathcal{X}} g(\boldsymbol{x}) = \min_{\boldsymbol{x} \in \mathcal{X}} \max_{\Delta\boldsymbol{x} \in \mathcal{U}} f(\boldsymbol{x} + \Delta\boldsymbol{x}) \tag{MM}$$

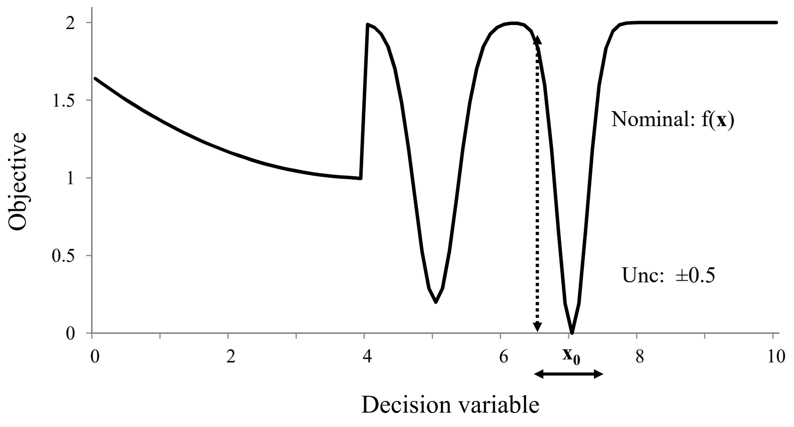
Finding the robust global optimum is based on an outer minimum worst case cost objective function value in $\mathcal{X}$, such that that minimum objective is based on an inner maximisation of the nominal objective function in the uncertainty neighbourhood around each solution $\boldsymbol{x} \in \mathcal{X}$.

Note that we assume that f is not restricted to $\mathcal{X}$, as $\boldsymbol{x} + \Delta\boldsymbol{x}$ may be outside of $\mathcal{X}$. Alternatively $\boldsymbol{x} + \Delta\boldsymbol{x} \in \mathcal{X}$ for all $\Delta\boldsymbol{x} \in \mathcal{U}$ could be achieved through a reduction in the size of the original $\mathcal{X}$ by Γ.

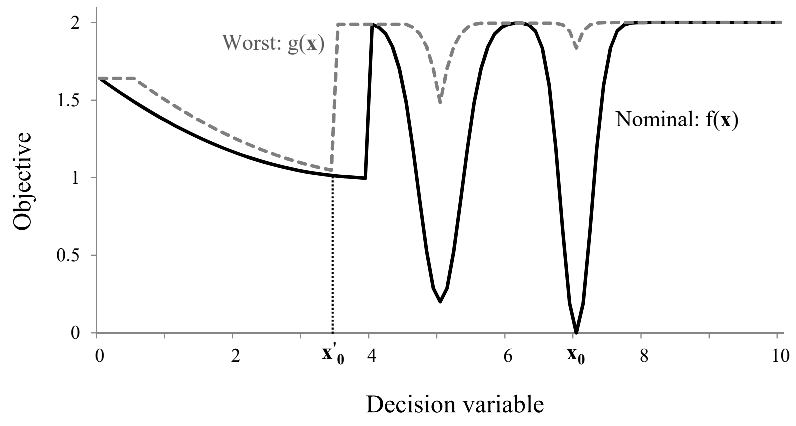
In Figure 3.1c the worst case cost $g(\boldsymbol{x})$ (dashed grey curve) at any individual point $\boldsymbol{x}$ can be determined by assessing the uncertainty neighbourhood around that point,



(a) The nominal global optimum is at $\boldsymbol{x}_0$.



(b) With uncertainty only a 'close' solution may be realised, which is of particular concern in the region of $\boldsymbol{x}_0$.



(c) The worst case cost curve (dashed grey) is generated by determining the maximum objective value in the uncertainty neighbourhood around all points $\boldsymbol{x}$ on the nominal (solid black) curve. Due to the uncertainty the global optimum shifts to $\boldsymbol{x}'_0$.

Figure 3.1: One dimensional problem due to [Kru12].

in order to identify the maximum value within that uncertainty neighbourhood. Then within the global minimisation search the nominal objective is superseded by the worst case cost. It can be observed that the global optimum has shifted to $\boldsymbol{x}_0'$. It should also be noted that if we were to ignore the implementation uncertainty and simply accept $\boldsymbol{x}_0$ as the global optimum, which is a common approach in practice, then we risk the possibility of a very poor outcome, i.e., $g(\boldsymbol{x}_0') < g(\boldsymbol{x}_0)$, whereas $f(\boldsymbol{x}_0') > f(\boldsymbol{x}_0)$.

Finally one additional assumption is made here for practical purposes when considering approaches to solving (MM), for example due to run-time considerations. This is the imposition of some limit (budget) on the number of model runs or function evaluations that can be undertaken.

3.1.1 Contributions and outline

In this paper we propose a particle swarm optimisation (PSO) framework encompassing a new robust metaheuristic capability for black-box problems under implementation uncertainty. The approach can be applied to general problems of reasonable dimension, where little if anything is known about the nature of the objective function surface, and under restrictions on the number of function evaluations (the budget). The framework developed here is based on an extension of PSO, see [KE95, KES01, Tal09]. Specifically we employ a PSO frame, augmenting it with adapted elements of the robust local search descent directions (d.d.) approach due to [BNT07, BNT10b, BNT10a], and the robust global largest empty hypersphere (LEH) approach due to [HGW19], as well as introducing original features. Whilst the resulting framework encompasses two new capabilities, due to the ability to employ each one independently our framework effectively has three alternative settings: the complete capability combining both features plus the option to switch off either of the new d.d. or LEH based sub-algorithms. We undertake a series of computational experiments comparing these new methods with a baseline robust PSO (rPSO), a global version of d.d. and LEH, see [BNT10b, HGW19]. We find that our new framework considerably outperforms these approaches on a large number of problem instances.

In Section 3.2 we review the current state of the art by discussing the relevant literature in Section 3.2.1 and details of the d.d. and LEH algorithms in Sections 3.2.2 and 3.2.3. We discuss PSO in Section 3.3, including an extension of the nominal formulation to a baseline rPSO formulation. In Section 3.4 we provide an illustrative example. We then outline our new robust framework in Section 3.5, including descriptions of our heuristic sub-algorithms. In Section 3.6 we describe the experimental set up used to test our new framework, and present our results. We end with a summary and consideration of potential further work in Section 3.7. The appendices cover descriptions of our experimental test functions, box plots of results, and a list of abbreviations.

3.2 State of the art

3.2.1 Literature review

The modern form of robust optimisation was first developed in [KY97] and [BTN98], since when the field has been strongly associated with mathematical programming through the use of appropriate uncertainty sets and the reformulation of mathematical programming problems, see e.g. the surveys [BBC11, GMT14, GS16]. In this setting the focus is on identifying a good formulation of the problem at hand, along with a tractable corresponding robust counterpart. This is not possible for black-box problems, where no knowledge of the problem structure is available.

In general, two types of uncertainty can be distinguished: parameter uncertainty, where the problem data is unknown; and implementation uncertainty, where the decision is subject to change during its implementation. In a mathematical programming context robust optimisation with parameter uncertainty has been widely applied to specific problems and formulations, while robust optimisation of black-box problems with implementation uncertainty is much less widely addressed, see [MWPL13, GS16, MWPL16].

A worst case analysis can be approached by applying standard global metaheuristics to both the inner maximisation and the outer minimisation. In a co-evolutionary approach inner and outer populations evolve separately, but the fitness of individuals in the outer minimisation is determined by individuals in an inner maximisation, see [Her99, SK02, Jen04, CSZ09, MKA11]. However a completely brute force approach using full inner maximisation searches to inform the outer minimisation involves large numbers of function evaluations, see [MWPL16]. Additional simplifications or assumptions are required to reduce the number of function evaluations in a co-evolutionary approach, see [CSZ09, MKA11].

An alternative robust evolutionary approach, introduced by [TG97], is based around the idea of 'genetic algorithms with a robust solution searching scheme' (GAs/RS^3) [BS07]. Uncertainty is added to the individuals in the population prior to the determination of the next generation; the next generation is then determined based on an assessment of the fitness of the extended (uncertain) population. [ONL06] adopts such an approach in a min max robust design analysis. This work also employs an approach that can be considered more generally in robust analyses, the use of emulation (surrogates or meta-models) alongside true objective function evaluations to reduce the potential burden of computational run times and the number of model-function evaluations, see [BS07, KVDHL16]. [ONL06] use surrogates for the inner optimisation local search. In [ZZ10] the inner maximisation is tackled using a surrogate-assisted evolutionary algorithm, whilst [MWPL13, uRLvK14, MWPL16, uRL17] all employ Kriging meta-modelling techniques. By contrast [CLSS17, SEFR19] employ Bayesian emulation approaches. [CLSS17] uses a mathematical programming approach assuming the availability of a valid Bayesian oracle, whilst [SEFR19] employ a Bayesian approach for very expensive-to-evaluate functions, applying it to test problems of up to 10 dimensions using only small numbers of function evaluations. However current emulation based approaches suffer from the same limitation, in that they struggle when applied to problems other than those of relatively low dimension.

Of particular interest here are the single-solution descent directions [BNT07, BNT10a, BNT10b] and largest empty hypersphere [HGW19] robust (min max) metaheuristics, and standard (i.e. not robust) population based metaheuristic, particle swarm optimisation [KE95, KES01, Tal09].

The d.d. approach is actually a robust local search. Given a start point in the decision variable space an inner maximisation is undertaken in the point's uncertainty neighbourhood. From this neighbourhood search undesirable 'high cost points' (hcps) are identified, and a direction which optimally points away from all of these hcps is determined by solving a quadratic program. A step is taken in this descent direction, to a new point where the process is repeated until no such direction can be found. This approach is considered in more detail in Section 3.2.2. This work also informs the global approach outlined in [BN10] where similar techniques are applied to the inner maximisation, but the outer minimization is tackled by simulated annealing. In [HGW19] d.d. has been simplistically extended to a global search through the use of random re-starts.

The LEH metaheuristic is a relatively new robust global approach which extends the idea of locally moving away from undesirable hcps to a global setting by identifying regions of the feasible region devoid of hcps and moving to the centres of such regions. Hence this approach is exploration-focussed. This approach is considered in more detail in Section 3.2.3.

The PSO approach is a population based metaheuristic inspired by swarm intelligence; the description given here is based on [KE95, KES01, Tal09]. A swarm consists of multiple particles, moving through the solution space. The position of each particle represents a point visited in the decision variable space, and the objective function value at that location. An additional attribute of a particle is its velocity, which here represents the vector (direction and step size) of the particle's movement. Based on some combination of an individual particle's own information and the collective information from other particles in the swarm, each particle moves to new locations as the algorithms iterates. It is the intention that what emerges from such complex, self-organising systems of particles approximates an efficient search of the solution space to identify global optima.

In terms of a robust PSO approach, [SK02, MKA11] consider two-swarm co-evolutionary PSO techniques while [HGW19] employs a baseline rPSO approach as a comparator test heuristic. [Dip10] develops several PSO formulations, and includes material on topologies, memory (archive) and test functions, however the approaches considered are essentially stochastic and only very low-dimensional problems are used in the testing. PSO and a baseline rPSO are considered in more detail in Section 3.3.

3.2.2 Local robust search using descent directions

Descent directions [BNT07, BNT10b, BNT10a] is a local search technique for solving the robust optimisation problem (MM), which uses the points evaluated in each inner maximisation local uncertainty neighbourhood analysis to inform a gradient descent approach. At a given point $\boldsymbol{x}$ an inner maximisation search is performed to approximate the worst case cost $\tilde{g}(\boldsymbol{x}) \approx g(\boldsymbol{x})$. In [BNT10b] an extensive two-stage gradient ascent search is employed for inner maximisations. All function evaluations are recorded in a history set H. From within the uncertainty neighbourhood $N(\boldsymbol{x}) = \{\boldsymbol{x} + \Delta\boldsymbol{x} \mid \Delta\boldsymbol{x} \in \mathcal{U}\}$ around a candidate point $\boldsymbol{x}$, the points with the greatest objective function values are identified as high cost points. The high cost set $H_\sigma(\boldsymbol{x})$ at any given point $\boldsymbol{x}$ is defined as:

$$H_\sigma(\boldsymbol{x}) := \{\boldsymbol{x}' \in H \cap N(\boldsymbol{x}) \mid f(\boldsymbol{x}') \geq \tilde{g}(\boldsymbol{x}) - \sigma\}$$

where σ is a threshold value for identifying hcps.

The intention is then to identify the descent direction $\boldsymbol{d}$ projecting from candidate point $\boldsymbol{x}^k$ at iteration k, which optimally points away from the points in $H_\sigma(\boldsymbol{x}^k)$. This is achieved by maximising the angle θ between $\boldsymbol{d}$ and the vectors connecting the points in $H_\sigma(\boldsymbol{x}^k)$ to $\boldsymbol{x}^k$. This is a second order cone problem and can be tackled using mathematical programming:

$$\begin{aligned}
&\min_{\boldsymbol{d},\beta} \ \beta && && \text{(Soc1)}\\
&\text{s.t.} \ \|\boldsymbol{d}\| \leq 1 && && \text{(Soc2)}\\
&\qquad \boldsymbol{d}^T\left(\frac{\boldsymbol{x}_i - \boldsymbol{x}^k}{\|\boldsymbol{x}_i - \boldsymbol{x}^k\|}\right) \leq \beta && \forall \boldsymbol{x}_i \in H_\sigma(\boldsymbol{x}^k) && \text{(Soc3)}\\
&\qquad \beta \leq -\varepsilon && && \text{(Soc4)}
\end{aligned}$$

Setting ε as a small positive scalar makes β negative in (Soc4). The left hand side of (Soc3) equates to $\|\boldsymbol{d}\| \cos\theta$, and is calculated for all points in $H_\sigma(\boldsymbol{x}^k)$. (Soc3) therefore states that β will correspond to the maximum value for $\cos\theta$ across all hcps. The objective (Soc1) is to minimise β. As β is negative the angle θ will be greater than 90^o and maximised. Finally, minimising β in combination with (Soc2) normalises $\boldsymbol{d}$.

The final component of the algorithm is the calculation of the step size to be taken once a descent direction $\boldsymbol{d}$ has been determined. At iteration k in the local search a step size ρ^k just large enough to ensure that all of the points in $H_\sigma(\boldsymbol{x})$ are at least on the boundary of the Γ-uncertainty neighbourhood of the next candidate solution at step

$k+1$ is used. We set $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \rho^k \boldsymbol{d}$, where ρ^k can be calculated using:

$$\rho^k = \min \left\{ \boldsymbol{d}^T(\boldsymbol{h} - \boldsymbol{x}^k) + \sqrt{(\boldsymbol{d}^T(\boldsymbol{h} - \boldsymbol{x}^k))^2 - \|\boldsymbol{h} - \boldsymbol{x}^k\|^2 + \Gamma^2} \mid \boldsymbol{h} \in H_\sigma(\boldsymbol{x}^k) \right\} \tag{Rho}$$

In the original formulation from [BNT10b] several loops are potentially applied in the algorithm, in order to try to identify a valid direction and to ensure that that direction is reasonable. The parameter σ is incrementally changed up to some limit, if (Soc) cannot be solved initially. Also given a valid direction vector $\boldsymbol{d}$, a further check is used to ensure that the step to be taken does not immediately encounter additional hcps from H beyond $N(\boldsymbol{x}^k)$.

This local stepping continues until no descent direction can be identified, and it is assumed that a robust local minimum has been reached. The approach can be extended to approximate a global search by randomly re-starting a new search each time the previous one completes. In [HGW19] this is employed within the constraint of a fixed budget of function evaluations.

3.2.3 Global robust search using largest empty hyperspheres

The largest empty hypersphere metaheuristic [HGW19] is a global method where the search progresses by moving to locations in the feasible region that are furthest away from all 'bad' points previously visited. Using the d.d. idea of identifying high cost points in a global sense, LEH uses a history set H of points evaluated and a high cost set H_τ which is a subset of H containing all points with nominal objective function value $f(\boldsymbol{x})$ greater than a threshold τ, which is set to the current estimated robust global minimum value. Note that with this notation, $H_\sigma(\boldsymbol{x}) = H_{\tilde{g}(\boldsymbol{x})-\sigma} \cap N(\boldsymbol{x})$.

Given H_τ, LEH uses a genetic algorithm (GA) to estimate a point $\boldsymbol{x}^k \in \mathcal{X}$ which is furthest from all hcps in H_τ. The search then moves to this point. This is repeated until the budget of available function evaluations is exhausted or no point $\boldsymbol{x}^k \in \mathcal{X}$ can be identified which is at least Γ away from all hcps. In either case the current estimate for the global robust minimum is accepted.

At each candidate point $\boldsymbol{x}^k$ an inner maximisation analysis is undertaken, however beyond the initial (random) start point each point in an inner maximisation analysis is compared to τ such that the Γ-radius uncertainty neighbourhood search can be stopped prematurely if any objective function value $f(\boldsymbol{x}^k + \Delta\boldsymbol{x}^k)$ is greater than τ. This is a recognition of the fact that the current point $\boldsymbol{x}^k$ cannot improve on the current estimate of the robust global optimum. This stopping condition potentially enables the LEH approach to explore $\mathcal{X}$ more efficiently.

3.3 Particle swarm optimisation

3.3.1 Motivation

The overarching motivation for our work is the development of improved robust metaheuristics for black-box problems under implementation uncertainty. Of particular interest are approaches that can be applied to general problems of moderate dimension, where run-time issues limit the numbers of function evaluations or model runs that can be undertaken, and no knowledge of the underlying problem or associated objective function surface is assumed. Whilst some work has been undertaken in this area, compared to optimisation without any consideration of uncertainty, stochastic optimisation, or robust optimisation for mathematical programming problems, this remains a less developed field.

Of the techniques currently applicable in this setting, the local d.d. [BNT07, BNT10b, BNT10a] and global LEH [HGW19] approaches offer considerable insight into some key issues. This is not least because whilst d.d. is locally exploitation focussed, LEH is exploration focussed in its targeting of regions of the decision variable space devoid of 'poor' points. That is these techniques span the exploitation versus exploration divide.

In particular these two approaches offer alternative stances on how to contend with what might be considered the additional 'burden' of a robust analysis under limitations on the number of function evaluations, i.e. the need to expend evaluations in the uncertainty neighbourhood analysis around individual points in the decision variable space in order to determine the robust value (the inner maximisation). Under budgetary restrictions we must therefore add the balancing of better estimating a candidate point's robust value versus the extent of the outer minimisation search, into the mix of exploration versus exploitation. This trade-off is complex, see for example [MLM15, EDHX17].

The d.d. approach explicitly uses the additional information gained from an uncertainty neighbourhood inner maximisation search to direct a local search, by identifying the direction that optimally points away from the worst neighbourhood points. By contrast a key component of the LEH approach is what is termed a 'stopping condition', that is the ability to terminate an inner maximisation search prior to completion due to a recognition that the current global robust best cannot be improved upon at a given candidate point. This has the potential to introduce significant efficiencies when expending function evaluations, thereby enabling a more extensive outer minimisation search. In fact we recognise the contrast in exploitation (d.d.) versus exploration (LEH) as an echo of the contrast between enhancing a search through the exploitation of the inner maximisation information (d.d.) versus attempting to limit inner maximisation searches in order to expend function evaluations more efficiently (LEH).

Here we are interested in the potential benefits of both the better use of the information gained from previous function evaluations, and of efficiency savings in terms of numbers of function evaluations. In particular we are interested in addressing these elements within a single framework. Compared to the individual-based d.d. and LEH techniques, we consider a population based approach more able to encompass these features under a single structure. In order to identify a suitable population based framework here, consideration must be given to the features necessary to enable both the use of the stopping condition component of LEH, and the use of uncertainty neighbourhood directional information generated by the calculation of some form of descent direct at a given candidate point.

The inclusion of a stopping condition requires that an inner maximisation search can be terminated early. Consider for example, how this might work in a fitness-based approach such as a genetic algorithm (GA). In order to effectively determine robust fitness at a candidate point, an inner maximisation must be complete. Early termination would not generate adequate fitness information: given a stopping threshold multiple members of the GA population could terminate their inner maximisation when an uncertainty neighbourhood point has been identified that exceeds that threshold, potentially leading to each being designated a similar fitness level close to the threshold objective function value. However by contrast if each inner maximisation were to complete, individual fitnesses could vary substantially. These two cases could lead to substantially different next generations due to the discrepancies in estimated fitnesses for members of the population. Therefore any such fitness-based approach does not suit a stopping condition of the kind under consideration here.

However considering a swarm-based approach such as PSO, each particle already has an in-built feature that can be exploited for stopping purposes, the best historic robust objective function value for each particle j. In the PSO case, stopping an inner maximisation prematurely if any Γ-radius uncertainty neighbourhood function evaluation exceeds that personal best threshold, τ^j, has no negative impact as movement in a standard PSO formulation is based on some combination of personal and neighbourhood best information. Neither of these pieces of information are affected by particle-level stopping. For a particle j, a function evaluation exceeding τ^j establishes that neither the historic best particle level information nor the current neighbourhood (e.g. global) best can be improved upon by the particle's current location. In such a situation terminating an inner search and moving on is appropriate, and desirable.

Furthermore, the primary component of the d.d. approach is the determination of a direction vector, therefore an approach that already uses a vector-based approach is best placed to accommodate further vector information. Individual particle level movement in a swarm-based approach such as PSO is vector-based. In addition, in considering how a d.d. vector and particle level stopping features might be incorporated into a PSO formulation, it can be recognised that both features might be incorporated into the same algorithm independently, thus allowing them to be considered – and their performance assessed – individually or in combination. Therefore, whilst amongst the substantial number of population based metaheuristics available, see e.g. [Tal09], there may be other suitable frameworks, a PSO approach clearly meets our requirements.

Here, therefore, we seek to develop an enhanced robust PSO-based framework, based on adapting key elements of the d.d. and LEH approaches, combined with novel features. To that end we first consider PSO in more detail.

3.3.2 Nominal PSO

There are many formulations of PSO, see [KES01, Kam09, ZWJ15, SBP18]. Here we describe one of the simplest, original formulations [KE95, KES01, Tal09]. This will form the basis of the robust framework to be developed here. We will first consider a 'standard', non-robust approach. A problem of the form (MM) can be considered in terms of its two constituent components: an inner maximisation and an outer minimisation. The PSO formulation described here should be appreciated as performing the outer minimisation component of (MM). We will return to the inner maximisation component when we discuss extending PSO to a baseline robust approach, rPSO.

PSO starts at iteration $t = 0$ with a population of N particles at randomly selected points $\boldsymbol{x}^j(0)$ in $\mathcal{X}$, where $j = 1, \ldots, N$. The function is evaluated at these points. For each particle the best position it has visited is designated $\boldsymbol{x}_*^j$, that is the position with the lowest objective function value $\tilde{g}(\boldsymbol{x}_*^j)$.

Particles are interconnected for information sharing, so each particle has an associated neighbourhood of other particles within which information can be shared. Different PSO formulations employ different neighbourhood strategies. Here we use global PSO as described in [SE98]. In global PSO the neighbourhood is the entire swarm and the information shared within the swarm is the global best position, that is the position $\hat{\boldsymbol{x}}_*$ in $\mathcal{X}$ with the lowest objective function value of all the points visited by all particles over all iterations.

From a particle's position $\boldsymbol{x}^j(t)$ at iteration t the particle's position is updated by the addition of its velocity vector $\boldsymbol{v}^j$:

$$\boldsymbol{x}^j(t) = \boldsymbol{x}^j(t-1) \; + \; \boldsymbol{v}^j(t) \tag{Move}$$

Following the recommendation of [Eng12] to initialize particle velocities at zero or at random values close to zero, here we consider the approach where each $\boldsymbol{v}^j(0)$ is separately initialised using uniform random sampling $\sim U(0\;,\;0.1)^n$. Beyond initialisation the following velocity formulation is used:

$$\boldsymbol{v}^j(t) = \omega \cdot \boldsymbol{v}^j(t-1) \; + \; C_1 \cdot \boldsymbol{r}_1 \cdot (\boldsymbol{x}_*^j - \boldsymbol{x}^j(t-1)) \; + \; C_2 \cdot \boldsymbol{r}_2 \cdot (\hat{\boldsymbol{x}}_* - \boldsymbol{x}^j(t-1)) \tag{Vel1}$$

Here $\boldsymbol{r}_1\;,\;\boldsymbol{r}_2 \sim U(0\;,\;1)^n$, that is each component of the random vectors $\boldsymbol{r}$ are randomly sampled individually, and multiplication between vectors is meant component wise. C_1, C_2 and ω are scalar terms. C_1 and C_2 represent 'learning' factors that weight the priority that a particle puts on its own (C_1) versus the global (C_2) historic success (that is over all iterations, to date). ω is an inertia term which moderates the effect of the preceding velocity on the current velocity.

As the particles move through $\mathcal{X}$ their individual $\boldsymbol{x}_*^j$ values and the global $\hat{\boldsymbol{x}}_*$ are updated as appropriate. If at any stage the next candidate position for any particle lies outside of the lower and upper bounds l_i and u_i of $\mathcal{X}$, here an invisible boundary condition is assumed, see [RR04]. Particles are allowed to leave the feasible region to naturally return to feasibility due to the pull of the $\boldsymbol{x}_*^j$ and $\hat{\boldsymbol{x}}_*$ information. Note that when a candidate moves outside of the feasible region no function evaluations are undertaken. Rather the velocity equation is updated by the particle's new location, with the $\boldsymbol{x}_*^j$ information remaining unchanged.

A standard PSO search can be extended to a baseline robust PSO search by adding an inner maximisation search component to the outer PSO minimisation search. Indeed it should be recognised that d.d. and LEH as described in Sections 3.2.2 and 3.2.3 focus on the outer minimisation component of the min max search (MM), so we will now give some consideration to inner maximisation.

3.3.3 Inner maximisation

The key requirement of any inner maximisation approach is the ability to accurately identify the maximum objective function value within the Γ-radius uncertainty neighbourhood around any given candidate point. However when dealing with real-world problems we must additionally take account of practical considerations. For simulation problems a common limiting feature is the number of model runs that can be performed, primarily due to simulation run times. In such a situation it is common to be restricted to an upper budget $B^{\max}$ on the number of model runs, which would in turn impact on the ability to accurately perform inner maximisation searches.

Where there are budgetary restrictions on the number of function evaluations, some trade-off must be achieved between the extent of each inner maximisation search (robustness) and the overall global search performance. However the trade-off between robustness and performance is not straightforward, see [MLM15, EDHX17]. In [BNT10b] the inner maximisation involves a series of two-stage gradient ascent searches within the Γ-uncertainty neighbourhood of a given candidate point, and assumes the availability of gradient information. Such an approach to the inner maximisation is comprehensive, but is in practice likely to prove prohibitive with increasing number of dimensions, even assuming the availability of gradient information.

In [HGW19] uniform random sampling is used in the Γ-radius hypersphere that forms the uncertainty neighbourhood around a candidate point, with the maximum value sampled taken as an approximation to the inner maximum. It is this approach that we adopt here for the inner maximisation in all heuristics considered.

The PSO framework introduced here comprises a main outer algorithm and three sub-algorithms. The inner maximisation sub-algorithm along with the outer minimisation frame constitute a baseline robust PSO metaheuristic, with additional d.d. and LEH sub-algorithms representing novel enhanced capabilities. Pseudo-code for inner maximisation by uniform random sampling in a Γ-radius hypersphere is shown in Algorithm 3. The outer minimisation (Algorithm 4), additional d.d. (Algorithm 5) and LEH (Algorithm 6) components are given subsequently. In the following, we do not explicitly list f, $\mathcal{X}$, or Γ as algorithm inputs, as they are always implied.

Algorithm 3 Γ-uncertainty neighbourhood inner maximisation inc. STOPPING option

```
    Input: $\boldsymbol{x}_c$, $B^{\max}$, $B^{\text{in}}$, $stopping$, $\tau$
 1: Calculate $f(\boldsymbol{x}_c)$ and store in $F_H$
 2: $H \leftarrow H \cup \{\boldsymbol{x}_c\}$
 3: $\tilde{g}(\boldsymbol{x}_c) \leftarrow f(\boldsymbol{x}_c)$
 4: $B^{\max} \leftarrow B^{\max} - 1$
 5: if ($stopping$) AND ($\tilde{g}(\boldsymbol{x}_c) > \tau$) then goto line 17 end if
 6: if ($B^{\max} = 0$) then break: goto end of Outer Min algorithm end if
 7: for all $i$ in $(1, \ldots, B^{\text{in}} - 1)$ do
 8:     Choose $\Delta\boldsymbol{x}_c^i \in \mathcal{U}$ uniformly at random
 9:     $\boldsymbol{x}_c^i \leftarrow \boldsymbol{x}_c + \Delta\boldsymbol{x}_c^i$
10:     Calculate $f(\boldsymbol{x}_c^i)$ and store in $F_H$
11:     $H \leftarrow H \cup \{\boldsymbol{x}_c^i\}$
12:     $\tilde{g}(\boldsymbol{x}_c) \leftarrow \max\{\tilde{g}(\boldsymbol{x}_c), f(\boldsymbol{x}_c^i)\}$
13:     $B^{\max} \leftarrow B^{\max} - 1$
14:     if ($stopping$) AND ($\tilde{g}(\boldsymbol{x}_c) > \tau$) then goto line 17 end if
15:     if ($B^{\max} = 0$) then break: goto end of Outer Min algorithm end if
16: end for
17: return $\tilde{g}(\boldsymbol{x}_c)$: estimated worst case cost at $\boldsymbol{x}_c$
```

Within any robust heuristic, given a candidate point $\boldsymbol{x}_c$ around which we want to perform a Γ-uncertainty neighbourhood inner maximisation, we call Algorithm 3. As input this requires the point information $\boldsymbol{x}_c$, a maximum number of function evaluations that can be undertaken in the entire search budget $B^{\max}$, the defined number of points to be evaluated within the inner maximisation analysis B^{in} (in the case of stopping this is the maximum number of points that could be evaluated), a boolean specifying whether or not the stopping condition is to be invoked *stopping*, and if required the stopping threshold τ.

It should be noted that B^{in} is a parameter that is tuned for all heuristics, within the experimental testing here, see Section 3.6.2. This supports the determination of an appropriate trade-off between the inner and outer searches, in the context of a budget on function evaluations.

The sub-algorithm starts by evaluating the function at the candidate point $\boldsymbol{x}_c$ (line 1), prior to moving on to uniformly randomly sample points in the Γ-uncertainty neighbourhood of $\boldsymbol{x}_c$ and evaluating the function at each of these points (lines 8 to 10). Function evaluations are recorded in the set F_H associated with H. When a function evaluation is performed the budget counter is reduced by 1 and a check is performed to ensure that $B^{\max}$ has not been exceeded. Note, however, that when the inner maximisation analysis has been prematurely ended due to $B^{\max}$ being exhausted, we do not want to return an estimate for $\tilde{g}(\boldsymbol{x}_c)$, but instead return to the end of the outer minimisation algorithm where the extant estimate for the robust global minimum is accepted (lines 6 and 15).

As the inner sampling proceeds the estimate for $\tilde{g}(\boldsymbol{x}_c)$ is updated as appropriate (lines 3 and 12). If the input value for *stopping* is TRUE and a function evaluation is detected which exceeds τ, the inner maximisation is terminated with the current estimate for $\tilde{g}(\boldsymbol{x}_c)$ returned (lines 5 and 14). Otherwise the full inner maximisation is completed, at which point the estimate for $\tilde{g}(\boldsymbol{x}_c)$ is returned to the outer minimisation sub-algorithm (line 17).

3.3.4 Baseline robust PSO

The easiest way to extend a PSO approach to an rPSO version and tackle the problem (MM) is to perform an inner maximisation search around any point in $\mathcal{X}$ visited by a particle, in order to replace the nominal objective function value $f(\boldsymbol{x})$ at any given point with the corresponding worst case cost value $\tilde{g}(\boldsymbol{x})$. With this approach the PSO formulation would remain unchanged. This is a baseline rPSO, which is used as the starting point for developing an enhanced rPSO metaheuristic framework here.

Pseudo-code for this baseline outer rPSO frame is given in Algorithm 4 for defined input parameter values N, C_1, C_2, ω, and B^{in}. In the experimental testing, and for all rPSO based heuristics, these five parameters are tuned, see Section 3.6.2. Note that we do not need to define the number of iterations over which the swarm is progressed, as this will be controlled by $B^{\max}$ within the inner maximisation Algorithm 3, which is called from line 13 of Algorithm 4. In addition we must input information for $B^{\max}$.

The PSO algorithm loops over iterations i until the budget $B^{\max}$ is exceeded (line 1), and over the N particles in the swarm (line 3). At the first iteration the particles are randomly initialised (line 6), but for subsequent iterations the particle positions, velocities, and personal best information are updated according to equations (Vel1) and (Move) (lines 8 and 9). Prior to performing any inner maximisation function evaluations the feasibility of the candidate point $\boldsymbol{x}^j(i)$ is confirmed. Here we use the boolean *ToBeEvaluated* (lines 4 and 10) to flag feasibility. Note that the use of the flag *ToBeEvaluated* is exploited further subsequently in the LEH capability sub-algorithm, Algorithm 6.

If $\boldsymbol{x}^j(i)$ is not in $\mathcal{X}$ the inner maximisation, and associated function evaluations, are skipped (line 12). This means that the particles personal best information $\boldsymbol{x}^j_*$ will not be updated, but otherwise the subsequent movement of particle $\boldsymbol{x}^j$ will continue according to equations (Vel1) and (Move).

Particle's $\boldsymbol{x}^j_*$ and the estimate of the robust global optimum $\hat{\boldsymbol{x}}_*$ are updated as appropriate (lines 14 and 15). At the end of the swarm-iterations loops the extant estimate of the robust global optimum $\hat{\boldsymbol{x}}_*$ is returned (line 20).

Algorithm 4 A baseline robust particle swarm optimisation algorithm

```
    Input: $B^{\max}$
    Parameters: $N$, $C_1$, $C_2$, $\omega$, $B^{\text{in}}$
 1: while ($B^{\max} > 0$) do
 2:     $i \leftarrow 0$
 3:     for all ($j$ in $1, \ldots, N$) do
 4:         $ToBeEvaluated \leftarrow$ TRUE
 5:         if ($i = 0$) then
 6:             Choose $\boldsymbol{x}^j(i) \in \mathcal{X}$ uniformly at random
 7:         else
 8:             Update particle velocity $\boldsymbol{v}^j(i)$ according to (Vel1)
 9:             Update particle position $\boldsymbol{x}^j(i)$ according to (Move)
10:             if ($\boldsymbol{x}^j(i) \notin \mathcal{X}$) then $ToBeEvaluated \leftarrow$ FALSE end if
11:         end if
12:         if ($ToBeEvaluated$) then
13:             $\tilde{g}(\boldsymbol{x}^j(i)) \leftarrow$ CALL Algorithm 3($\boldsymbol{x}^j(i)$, $B^{\max}$, $B^{\text{in}}$, FALSE, 0)
14:             if ($i = 0$) OR ($\tilde{g}(\boldsymbol{x}^j(i)) < \tilde{g}(\boldsymbol{x}^j_*)$) then $\boldsymbol{x}^j_* \leftarrow \boldsymbol{x}^j(i)$ end if
15:             if ($i = 0$ AND $j = 0$) OR ($\tilde{g}(\boldsymbol{x}^j(i)) < \tilde{g}(\hat{\boldsymbol{x}}_*)$) then $\hat{\boldsymbol{x}}_* \leftarrow \boldsymbol{x}^j(i)$ end if
16:         end if
17:     end for
18:     $i \leftarrow i + 1$
19: end while
20: return A robust solution $\hat{\boldsymbol{x}}_*$
```

3.4 Comparison of baseline heuristics

When it comes to the testing of new heuristics in Section 3.6 we require comparator robust heuristics against which to assess performance. Here we use the three baseline approaches already discussed: re-starting d.d., LEH and the baseline rPSO. In order to give some indication of the different natures of the searches due to each comparator robust metaheuristic considered we introduce a two-dimensional problem and plot exemplar searches due to each heuristic.

Consider one of the test problems to be used in our experimental test suite Section 3.6.1, the multi-dimensional Pickelhaube problem. A full description of this function is given in Appendix 3.8.2, and plots of the nominal and worst case (Γ=1) 2D Pickelhaube problem are shown in Figures 3.5i and 3.5j. In our formulation for the 2D problem the nominal global optimum is at $(-35, -35)$, whilst the robust global optimum is at $(-25, -25)$.

Contour plots of individual example searches for each of the three baseline approaches applied to this 2D problem are shown in Figure 3.2. These plot should be seen as indicative exemplars. Plots on the left indicate all points evaluated and the underlying contour is the nominal plot. Plots on the right show the improving search path of the currently identified global robust optima, over the underlying worst case contour.

For the d.d. search in Figures 3.2a and 3.2b the inner maximisation groupings of evaluated points can be seen to follow a series of paths, each towards a robust local optimum and based on a series random re-starts at the completion of the previous local search. It can be seen that two such local searches successfully hone in on the robust global optimum.

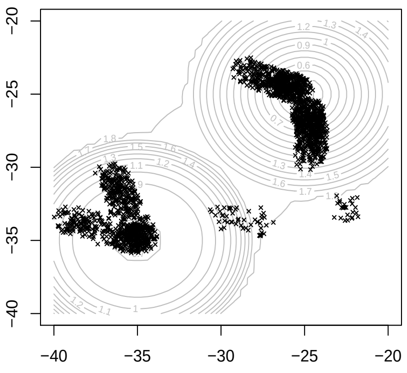
The nature of the LEH search in Figures 3.2c and 3.2d is very different to the d.d. search, and two obvious features are apparent. First is the extensive exploration of the solution space as the search jumps to centres of regions devoid of poor points. Second is that in many cases the inner maximisation groupings of evaluated points are sparse, and in fact are often just single point evaluations as the stopping condition recognises that the current location cannot improve on the existing estimate of a global robust optima – and so the algorithm immediately moves on without undertaking a full Γ-uncertainty neighbourhood search. The location of the robust global optima is successfully identified.

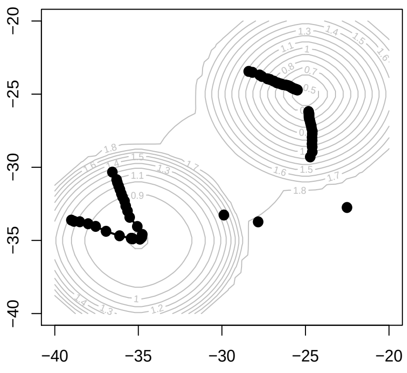
Finally for the baseline rPSO search in Figures 3.2e and 3.2f, the combined exploratory-exploitation nature of a PSO search can be observed. The approximate location of each particle within each iteration is identifiable by the cluster of points evaluated within that particle's Γ-uncertainty neighbourhood. Again the location of the robust global optima is successfully identified.

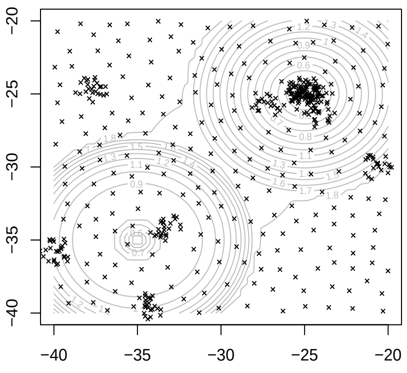
3.5 Enhanced robust particle swarm optimisation

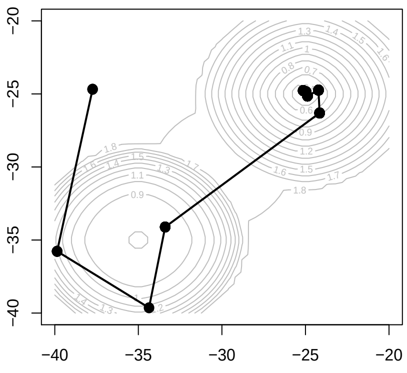
3.5.1 New capabilities

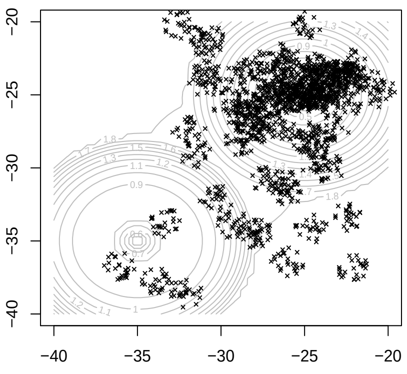
In both d.d. and LEH approaches the additional function evaluations required to calculate robust, as opposed to nominal, values are used to direct the search. In the case of d.d. this points the search towards the optimal local direction to avoid hcps, whilst for LEH this points the search towards the location which is globally furthest from all existing hcps. Here we seek to exploit both of these local and global search directions.


(a) d.d. points


(b) d.d. current best


(c) LEH points


(d) LEH current best


(e) rPSO points

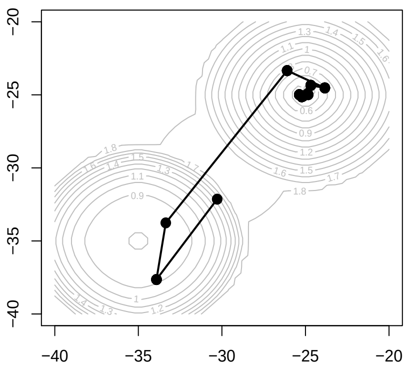

(f) rPSO current best

Figure 3.2: Example searches of the 2D version of the Pickelhaube problem (see Figures 3.5i and 3.5j) with $\Gamma = 1$, for the baseline metaheuristics. Evaluated points are shown on the left, with the path of the improving current robust best on the right. The outer heuristics are (top to bottom): d.d. with re-start, LEH, and robust PSO. The nominal global optimum is at $(-35, -35)$, and the robust global optimum is at $\sim(-25, -25)$.

The rPSO framework introduced here allows for the d.d. and LEH enhancements to be employed in combination, or individually. This framework employs a baseline rPSO comprising the outer and inner layers described in Algorithms 4 and 3 respectively, in conjunction with new sub-algorithms to execute the features described below.

3.5.1.1 rPSO d.d. capability

The basic PSO movement formulation involves a weighted combination of two direction vectors, added to a particle's current position, generating the movement to a new location. The two weighted direction vectors are based on the differences between a particle's current location and its historic personal best and, here, the current estimated global best. An obvious approach, given a third piece of vector information – a local uncertainty neighbourhood d.d. vector – is to simply add the weighted d.d. vector to the original vector calculation. Here each d.d. vector can be calculated at the particle level at each candidate point, given an inner maximisation at that point. Furthermore parameter tuning including all three weighting parameters will enable identification of the best combination of vector information.

3.5.1.2 rPSO LEH capability

In the LEH formulation at each new candidate point an inner maximisation begins but can terminate early if an uncertainty neighbourhood point exceeds the currently estimated robust global minimum. At its most efficient this may frequently mean only a single function evaluation is undertaken at a candidate, if it is immediately determined that that evaluation exceeds the global best. Incorporating this into PSO at a particle level, using particle best information as a stopping threshold, will introduce the desired inner maximisation efficiency savings. However this can be taken further. For LEH in a restricted budget on function evaluations setting, the historic record of function evaluations is unlikely to include any points in the uncertainty neighbourhood of a candidate location, as the very nature of LEH is to move to previously unexplored regions. In a population based approach however this may not be the case, and in fact it may be desirable that there is some convergence of members of the population in the decision variable space. This introduces the potential that no function evaluations may need to be undertaken at an individual particle's candidate location if the historic record identifies a previously evaluated point in the candidate uncertainty neighbourhood with objective function value greater than the particle's threshold. This is an additional efficiency, but also opens up the possibility of a further feature.

The non-requirement to perform even a single function evaluation at an individual particle's current candidate location allows for the consideration of a particle becoming 'dormant'. Not requiring any function evaluations at a location could be due to moving into an already visited region of the decision variable space and identifying neighbouring points in the historic record exceeding the particle's threshold, as discussed above. However alternatively this could be due to the particle moving outside of the feasible region. That is the invisible boundary condition [RR04] employed here also means that individual particles may become dormant in the sense that they move outside of the feasible region, necessitating no function evaluations and under the expectation of subsequent movements ultimately returning the particle to feasibility, driven by personal and neighbourhood best information.

If either of the situations described above were to repeat over several swarm iterations it would be reasonable to consider an approach that interrupts individual particles that have become dormant: 'stuck' either in previously visited or infeasible areas. By introducing a dormancy threshold representing the number of iterations before a particle is deemed truly dormant and requiring further action, that dormancy threshold can be parameter-tuned.

Given a particle's dormancy it seems reasonable to consider some action, so here we introduce an exploratory component. Taking the other main element of the LEH approach, the calculation of the largest empty region devoid of all previously evaluated points with objective value greater than some defined threshold, we relocate-reinitialise the particle at the centre of that LEH. Here we set the 'high cost' threshold, identifying which points from the historical record of function evaluations to avoid in the calculation of the largest empty region, equal to the current robust global minimum.

3.5.2 Enhanced capability: d.d. sub-algorithm

We start by considering the use of d.d. information to enhance the PSO velocity equation (Vel1) through the addition of a further velocity component of the form:

$$C_3 \cdot \boldsymbol{r}_3 \cdot \boldsymbol{dd}^j(t-1) \tag{ddVel}$$

For each particle j at each iteration t we perform a single-step d.d. calculation using the uncertainty neighbourhood points around the particle's position. Given a valid unit length direction vector $\boldsymbol{d}^j(t-1)$ as a result of solving (Soc), a final vector $\boldsymbol{dd}^j(t-1)$ is calculated by scaling using the term $\rho^j(t-1)$, which is calculated as in equation (Rho) in Section 3.2.2:

$$\boldsymbol{dd}^j(t-1) = \rho^j(t-1) \cdot \boldsymbol{d}^j(t-1)$$

If no such direction can be calculated then $\boldsymbol{dd}^j(t-1)$ is set to the zero vector. Incorporating this additional velocity component into the original velocity formulation (Vel1), beyond the initialisation of the particles at iteration 0 the following velocity formulation is used:

$$\boldsymbol{v}^j(t) = \omega\cdot\boldsymbol{v}^j(t-1)+C_1\cdot\boldsymbol{r}_1\cdot(\boldsymbol{x}^j_*-\boldsymbol{x}^j(t-1))+C_2\cdot\boldsymbol{r}_2\cdot(\hat{\boldsymbol{x}}_*-\boldsymbol{x}^j(t-1)+C_3\cdot\boldsymbol{r}_3\cdot\boldsymbol{dd}^j(t-1) \tag{Vel2}$$

Pseudo-code for the calculation of the additional d.d. velocity component (ddVel) is given in Algorithm 5. In terms of the overarching framework, Algorithm 4 is still valid but with a single change: in line 8 the enhanced velocity equation (Vel2) now replaces equation (Vel1).

Algorithm 5 requires input information for the candidate point of interest $\boldsymbol{x}_c$, the history set H and the associated set of function evaluation values F_H. In addition d.d. calculation parameter values are required: the initial σ value, the lowest value this can take σ_{limit}, and the number of reduction-steps which can be applied in reducing σ from its initial value down to σ_{limit}, in repeated attempts to solve (Soc) when the previous attempts have failed, σ_{no} – see Section 3.2.2. Also a factor, $init\rho_{Min}$, is required to ensure that any calculated d.d. vector achieves a minimum size. Finally the new input scalar parameter value C_3 is required for the final calculation of the $\boldsymbol{dd}_c$ vector.

In the experimental testing the σ, σ_{limit}, $init\rho_{Min}$ and C_3 parameters are tuned, see Section 3.6.2. This is in addition to the tuned baseline rPSO parameters, see Section 3.3.4. σ_{no} could be tuned but is pre-set in the experimental testing here in order to better control heuristic run times.

The sub-algorithm begins with a feasibility check for the input candidate point $\boldsymbol{x}_c$ (line 2). If no function evaluations are undertaken at a given candidate point due to it lying outside of $\mathcal{X}$ (line 18) then we instead set $\boldsymbol{dd}^j(t-1)$ to include only values in the dimensions which are infeasible (in all feasible dimensions the vector component retains the initialisation setting of 0, see line 1). The non-zero vector components are all set to magnitude Γ, with the sign for each dimension determined in order to point back into the feasible region (lines 19 to 25). As with all d.d. vectors this is then multiplied by a scalar C_3 value and a random vector $\boldsymbol{r_3}$ (line 26). The intention here is to promote a return to $\mathcal{X}$, beyond the existing draw of a particle's $\boldsymbol{x}^j_*$ and the global $\hat{\boldsymbol{x}}_*$ information (which are both in $\mathcal{X}$).

The high cost set H_σ and associated set of function evaluation values F_{H_σ} are initialised (line 3). Next is the attempt to solve the second order cone problem (Soc) and identify a valid descent direction based on the high cost set H_σ, see lines 5 to 16. As this is a mathematical programming problem in practice this is achieved by a call to an optimisation software package.

As described in Section 3.2.2, if (Soc) cannot be solved immediately it may be retried multiple times with reducing values of σ and hence with H_σ containing fewer points (lines 13 and 14). Flagging of the solution to (Soc) is controlled by the boolean $SolvedSOCP$. If (Soc) is solved the original normalised direction vector is re-scaled according to equation (Rho) (line 9) or the defined minimum distance ($init\rho_{Min} * \Gamma$) (line 10), prior to the rPSO velocity equation update due to equation (ddVel) (line 17). If ultimately (Soc) cannot be solved the initialisation setting of $\boldsymbol{dd}_c$ to $\mathbf{0}$ is retained (line 1).

Contour plots of an example search using the framework operating with only the enhanced d.d. capability applied to the 2D Pickelhaube problem are shown in Figures 3.3a and 3.3b on page 94, to give an indication of the nature of an enhanced search. In

Algorithm 5 Calculating the additional d.d. velocity component (ddVel)

```
    Input: $\boldsymbol{x}_c$, $H$, $F_H$
    Parameters: $\sigma$, $\sigma_{limit}$, $\sigma_{no}$, $init\rho_{Min}$, $C_3$

 1: $\boldsymbol{dd}_c \leftarrow \mathbf{0}$
 2: if ($\boldsymbol{x}_c \in \mathcal{X}$) then
 3:     Initialise $H_\sigma$ and $F_{H_\sigma}$
 4:     $SolvedSOCP \leftarrow$ FALSE
 5:     while ($!SolvedSOCP$) AND ($\sigma \geq \sigma_{limit}$) do
 6:         Try: $\boldsymbol{d}_c \leftarrow$ Solve (Soc) for $\boldsymbol{x}_c$ and $H_\sigma$
 7:         if (Solve (Soc) is successful) then $SolvedSOCP \leftarrow$ TRUE end if
 8:         if ($SolvedSOCP$) then
 9:             Calculate $\rho$ according to (Rho)
10:             if ($\rho < (init\rho_{Min} * \Gamma)$) then $\rho \leftarrow (init\rho_{Min} * \Gamma)$ end if
11:             $\boldsymbol{d}_c \leftarrow \rho \cdot \boldsymbol{d}_c$
12:         else
13:             $\sigma \leftarrow \sigma - (\sigma - \sigma_{limit})/\sigma_{no}$
14:             Update $H_\sigma$ and the associated $F_{H_\sigma}$
15:         end if
16:     end while
17:     if ($SolvedSOCP$) then $\boldsymbol{dd}_c \leftarrow C_3 \cdot \boldsymbol{r_3} \cdot \boldsymbol{d}_c$ end if
18: else
19:     for all ($i$ in $1, \ldots, n$) do
20:         if ($(\boldsymbol{x}_c)_i \leq l_i$) then
21:             $(\boldsymbol{dd}_c)_i \leftarrow \Gamma$
22:         else if ($(\boldsymbol{x}_c)_i \geq u_i$) then
23:             $(\boldsymbol{dd}_c)_i \leftarrow -\Gamma$
24:         end if
25:     end for
26:     $\boldsymbol{dd}_c \leftarrow C_3 \cdot \boldsymbol{r}_3 \cdot \boldsymbol{dd}_c$
27: end if
28: return Additional d.d. velocity component $\boldsymbol{dd}_c$
```

these plots and subsequently when discussing results for this capability we will use the nomenclature ‘rPSOdd’. Unsurprisingly the nature of these plots is somewhat similar to the baseline rPSO search seen in Figures 3.2e and 3.2f. However in addition the new Γ-uncertainty neighbourhood descent directions component in the velocity function, for each particle at each location, does appear to add elements of robust local search at the particle level. The extent of any d.d. component will be heavily influenced by the d.d. sub-algorithm parameter value settings.

3.5.3 Enhanced capability: LEH sub-algorithm

Our second enhancement to the rPSO formulation involves augmenting the baseline rPSO Algorithm 4 with an additional sub-algorithm to perform elements of the LEH approach due to [HGW19]. Here we incorporate two LEH-based components into our enhanced capability: the stopping condition, and the calculation of the largest empty hypersphere devoid of high cost points and placement of candidates at the centre of the calculated LEH. This further leads to an increased role for the use of the historic function evaluation information from the history set H.

We have already included the stopping condition when developing the Γ-uncertainty neighbourhood inner maximisation Algorithm 3. That sub-algorithm is set up to accept boolean input information *stopping* to flag whether or not the inner maximisation stopping condition should be invoked, plus the associated stopping threshold value τ if *stopping* is TRUE. Whereas in the baseline rPSO Algorithm 4 *stopping* was set to FALSE (Algorithm 4 line 13), here the stopping condition is invoked at the particle level by setting *stopping* to TRUE and using the candidate particle j personal best value $\tilde{g}(\boldsymbol{x}_*^j)$ for the stopping threshold τ. So for the LEH sub-algorithm (Algorithm 6), in line 13 of the outer minimisation (Algorithm 4) the call to the inner maximisation (Algorithm 3) becomes:

$$\tilde{g}(\boldsymbol{x}^j(i)) \leftarrow \textbf{CALL}\ \textit{Algorithm}\ 3\ (\boldsymbol{x}^j(i),\ B^{\max},\ B^{\text{in}},\ \textit{TRUE},\ \tilde{g}(\boldsymbol{x}_*^j)) \qquad \text{(call 3)}$$

In addition, however, within Algorithm 6, we introduce a pre-inner maximisation check of the history set H. Again this sub-algorithm requires input information for the candidate point of interest $\boldsymbol{x}_c$, and the history set H and associated set F_H. Further, Algorithm 6 needs to access the full particle information for particle $j(i)$ associated with the current candidate point $\boldsymbol{x}_c$, and the global best value $\tilde{g}(\hat{\boldsymbol{x}}_*)$. Also Algorithm 6 uses the counter $\textit{dormancy}^j_{\textit{count}}$ and the parameters *ToBeEvaluated* and $\textit{dormancy}_{\textit{limit}}$; the latter is tuned in our experiments along with the baseline rPSO parameters, Section 3.6.2. A further parameter $\textit{placement}_{\textit{limit}}$ is also introduced here but is not tuned, and instead pre-set to control heuristic run times. $\textit{dormancy}^j_{\textit{count}}$, $\textit{dormancy}_{\textit{limit}}$ and $\textit{placement}_{\textit{limit}}$ are described below.

The pre-inner maximisation check of H is to identify if there are existing points in the uncertainty neighbourhood of the candidate location $\boldsymbol{x}_c$. If there are, we then check if any such points already have nominal objective function value $f(\boldsymbol{x})$ greater than the particle threshold $\tilde{g}(\boldsymbol{x}_*^j)$. If so we determine that we do not need to perform any inner maximisation search for candidate location $\boldsymbol{x}_c$. We have already seen the use of boolean *ToBeEvaluated* to flag the need to undertake inner maximisation function evaluations, due to feasibility issues, in Algorithm 4. Here we extend the use of *ToBeEvaluated* to also flag when there is no need to undertake inner maximisation function evaluations due to the the pre-inner maximisation check (lines 1 to 4). In a similar fashion to the case of $\boldsymbol{x}_c$ being infeasible the particle j velocity information is updated by the particle j location $\boldsymbol{x}_c$, but $\boldsymbol{x}_*^j$ remains unchanged.

Algorithm 6 Re-locating particles using elements of the LEH heuristic

```
    Input: Particle $j(i)$, $\boldsymbol{x}_c$, $H$, $F_H$, $\hat{\boldsymbol{x}}_*$
    Parameters: $ToBeEvaluated$, $dormancy^j_{count}$, $dormancy_{limit}$, $placement_{limit}$
 1: if $(\boldsymbol{x}_c \notin \mathcal{X})$ OR $(\max\{\tilde{g}(\boldsymbol{h}) \mid \boldsymbol{h} \in N(\boldsymbol{x}_c)\} > \tilde{g}(\boldsymbol{x}_*^j))$ then
 2:     $ToBeEvaluated \leftarrow$ FALSE
 3:     $dormancy^j_{count} \leftarrow dormancy^j_{count} + 1$
 4: end if
 5: if $(dormancy^j_{count} > dormancy_{limit})$ then
 6:     $lehComplete \leftarrow$ FALSE
 7:     $countLEHtry \leftarrow 0$
 8:     while $(!lehComplete)$ AND $(countLEHtry < placement_{limit})$ do
 9:         $\boldsymbol{p} \leftarrow$ solution to (lehMM)
10:         Calculate $f(\boldsymbol{p})$ and store in $F_H$
11:         $H \leftarrow H \cup \{\boldsymbol{p}\}$
12:         $countLEHtry \leftarrow countLEHtry + 1$
13:         if $(f(\boldsymbol{p}) < \tilde{g}(\hat{\boldsymbol{x}}_*))$ then $lehComplete \leftarrow$ TRUE end if
14:     end while
15:     Re-set all particle $j(i)$ details to initialisation values
16:     $dormancy^j_{count} \leftarrow 0$
17:     $\boldsymbol{x}^j(i) \leftarrow \boldsymbol{p}$
18: else
19:     Update particle position $\boldsymbol{x}^j(i)$ according to (Move)
20: end if
21: return $ToBeEvaluated$ and input particle $j(i)$ updated if appropriate
```

The second feature of LEH that we exploit here is the exploration-based locating of new candidates at points furthest away from all previously visited ‘bad’ high cost points – which equates to placing candidates at the centre of the LEH (empty of hcps). Here we apply such an approach to relocate individual particles j based on a determination that particle j is ‘dormant’. Dormancy is based on a count of the number of swarm iterations over which no function evaluations have been performed for particle j, which may either be due to an infeasible candidate location, or due to the pre-inner maximisation check described (lines 1 to 4). To this end we introduce a particle level count of the number of dormant iterations $dormancy^j_{count}$, and the dormancy count level which triggers the relocation of a particle $dormancy_{limit}$. The counter is incremented as appropriate (line 3).

For a particle j, given the exceeding of the dormancy count level (line 5), an LEH calculation is undertaken to re-position particle j to the centre of the identified LEH. If, however, particle j is not identified as being dormant, the original update rule for the particle is used (line 19).

In the original LEH algorithm, see Section 3.2.3, the high cost set is defined as H_τ, a subset of the history set H containing all points with nominal objective function value $f(\boldsymbol{x})$ greater than a threshold τ. Within that algorithm the same threshold τ is employed for the stopping condition and for the identification of an LEH. Here we employ $\tau = \tilde{g}(\hat{\boldsymbol{x}}_*)$ with the intention of trying to re-locate particle j away from points that we already know cannot improve on our current estimate for the robust global minimum $\hat{\boldsymbol{x}}_*$.

We seek to estimate the point $\boldsymbol{p} \in \mathcal{X}$ furthest from all designated high cost points $\boldsymbol{h} \in H_{\tilde{g}(\hat{\boldsymbol{x}}_*)}$. This is the max min problem:

$$\max_{\boldsymbol{p} \in \mathcal{X}} \min_{\boldsymbol{h} \in H_{\tilde{g}(\hat{\boldsymbol{x}}_*)}} \|\boldsymbol{p} - \boldsymbol{h}\|, \tag{lehMM}$$

where $\|\cdot\|$ is the Euclidean norm.

We use the approach due to [HGW19] to estimate the solution to (lehMM), employing a genetic algorithm (line 9). As is the case for the baseline LEH comparator heuristic, within our experimental testing the parameters controlling the GA applied to solving problem (lehMM) are tuned, see Section 3.6.2.

There is a final element of this particle relocation. In a further attempt to enhance the exploratory nature of the rPSOleh heuristic, at the potential new candidate point $\boldsymbol{p}$ we perform a single point function evaluation (line 10). If this value is less than $\tilde{g}(\hat{\boldsymbol{x}}_*)$ we accept the new candidate point (line 13), however if not we perform further LEH calculations up to some input number of times $placement_{limit}$ that this retry can occur (lines 8 to 14). This process is controlled by a counter, lines 7 and 12, and success flag in line 6. If the number of retries is exhausted the final potential candidate $\boldsymbol{p}$ is accepted. Note that with each LEH calculation an additional point is added to H, impacting subsequent LEH calculations.

As this LEH calculation effectively re-initialises particle j, the previous particle j information including initial velocity and $\boldsymbol{x}^j_*$ need to be re-set to the particle initialisation settings, overwriting the existing information (lines 15 and 16). Subsequently new particle information will be established as the outer minimisation search (Algorithm 4) progresses. Particle j is then re-located to $\boldsymbol{p}$ (line 17). This relocation could happen to the same particle j more than once over the course of the heuristic search.

The LEH sub-algorithm (Algorithm 6) can then be accessed from the baseline rPSO Algorithm 4 by replacing lines 9 and 10 there with the single line reference to Algorithm 6:

$$\text{Update particle position } \boldsymbol{x}^j(i) \text{ according to Algorithm 6} \qquad \text{(relocateLEH)}$$

In addition, to use Algorithm 4 $dormancy^j_{count}$ values need to be introduced and initialised (set to zero) in Algorithm 4. The addition of a line within the IF statement, lines 5 to 7, achieves this:

$$dormancy^j_{count} \leftarrow 0 \qquad \text{(addCounter)}$$

Also $dormancy_{limit}$ and $placement_{limit}$ would need to be defined as additional input parameters in Algorithm 4.

Contour plots of an example search using the framework operating with only the enhanced LEH capability applied to the 2D Pickelhaube problem are shown in Figures 3.3c and 3.3d. In these plots and subsequently when discussing results for this capability we will use the nomenclature 'rPSOleh'. The nature of these plots is primarily a combination of the example rPSO and LEH searches seen in Figure 3.2. There are some inner maximisation groupings of evaluated points for each particle, as the particles iterate, plus some limiting of the extent of these inner searches – moving on without undertaking a full Γ-uncertainty neighbourhood search, plus the extensive LEH exploration of the solution space as the search moves to regions devoid of hcps.

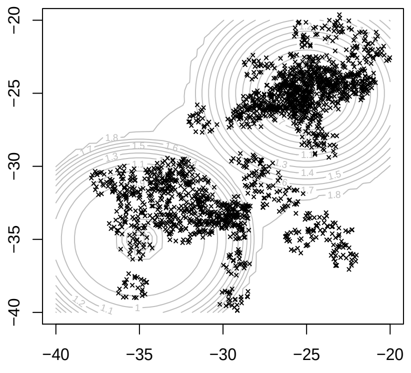
3.5.4 Full enhanced capability

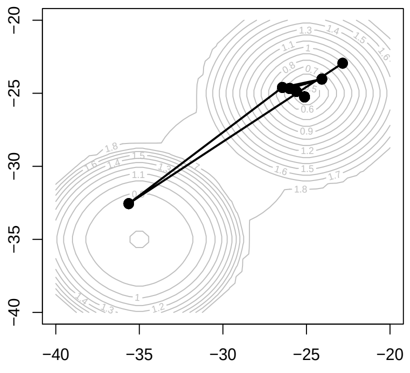
The operation of the full enhanced capability within our rPSO framework employs both of the two new sub-algorithms: we augment the baseline outer minimisation (Algorithm 4) and inner maximisation (Algorithm 3) with Algorithms 5 and 6. In combination the new sub-algorithms require input information for $\boldsymbol{x}_c$, H, F_H, full particle information for particle $j(i)$, $\hat{\boldsymbol{x}}_*$ and $dormancy^j_{count}$. Parameter values for σ, σ_{limit}, σ_{no}, $init\rho_{Min}$, C_3, $ToBeEvaluated$, $dormancy_{limit}$ and $placement_{limit}$ are also required.

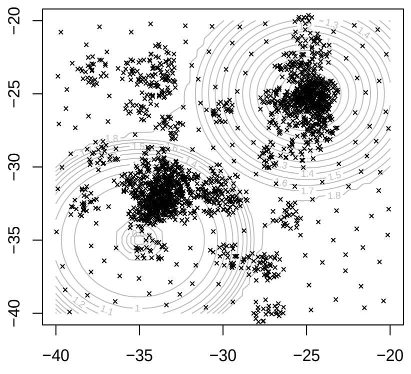
In the experimental testing the σ, σ_{limit}, $init\rho_{Min}$, C_3 and $dormancy_{limit}$ parameters are tuned, see Section 3.6.2, in addition to the tuned baseline rPSO parameters (Section 3.3.4) and the parameters controlling the GA applied to solving problem (lehMM) (Section 3.5.3).

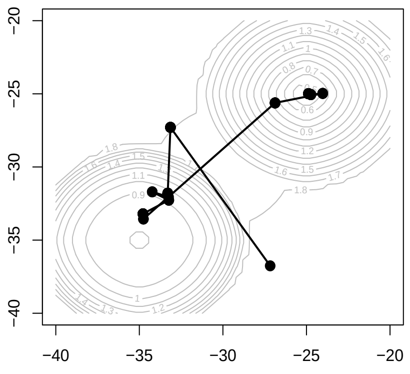
Algorithm 4 still requires a single change to accommodate the d.d. component: in line 8 the enhanced velocity equation (Vel2) replaces equation (Vel1). To accommodate the LEH components Algorithm 4 requires several minor updates. The call to the inner maximisation Algorithm 3 must be updated in accordance with (call 3). Plus the updates to include (relocateLEH) and (addCounter), and the addition of input parameters $dormancy_{limit}$ and $placement_{limit}$ to Algorithm 4, as described in Section 3.5.3.

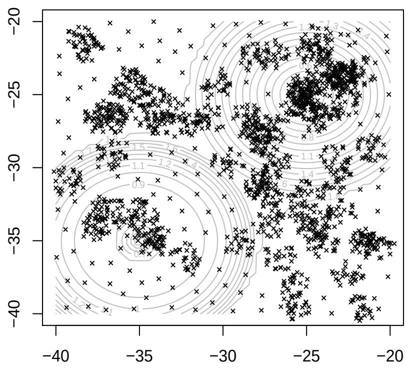
These updates and calls to Algorithms 3, 5 and 6 from the baseline rPSO Algorithm 4 complete the full capability within our rPSO framework.


(a) rPSOdd points


(b) rPSOdd current best


(c) rPSOleh points


(d) rPSOleh current best


(e) rPSOlehdd points

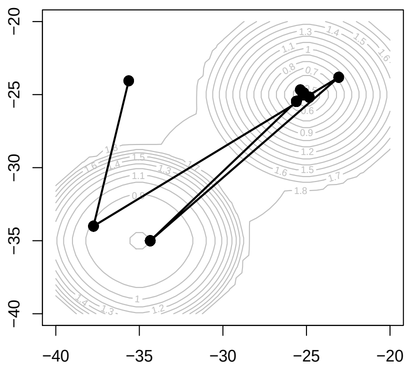

(f) rPSOlehdd current best

Figure 3.3: Example searches of the 2D version of the Pickelhaube problem Figures 3.5i and 3.5j with Γ=1, for the new rPSO framework. Evaluated points are shown on the left, with the path of the improving current robust best on the right. The outer framework settings are: enhanced d.d capability only (rPSOdd), enhanced LEH capability only (rPSOleh), and full enhanced capability (rPSOlehdd). The nominal global optimum is at $(-35, -35)$, and the robust global optimum is at $\sim(-25, -25)$.

Contour plots of an example search using the full capability applied to the 2D Pickelhaube problem are shown in Figures 3.3e and 3.3f. In these plots and subsequently when discussing results for this capability we will use the nomenclature 'rPSOlehdd'. The nature of these plots is somewhat similar to the plots combining just the PSO and LEH search elements, shown in Figures 3.3c and 3.3d. Also, however, the additional particle level d.d. component in the velocity function appears to introduce some elements of robust local search, although the extent of any such d.d. component will be influenced by the d.d. sub-algorithm parameter value settings.

3.6 Computational experiments

3.6.1 Experimental set up

In Section 3.4 we introduced three global robust metaheuristics as a baseline against which to test our three new approaches:

1. Baseline: largest empty hypersphere (LEH) [HGW19].
2. Baseline: repeating descent direction (d.d.) based on [BNT10b].
3. Baseline: robust particle swarm optimisation (rPSO) based on 'standard' PSO formulation [KE95, KES01, Tal09].

Our new framework allows for three alternative settings: the complete capability (rPSOlehdd), the use of the enhanced d.d. capability alone (rPSOdd), or the use of the enhanced LEH capability alone (rPSOleh). This enables us to generate three sets of new results for consideration against the three comparators.

Recall that all inner maximisation analysis is undertaken exclusively using uniform random sampling in the Γ-radius hypersphere that forms the uncertainty neighbourhood of any given point, see Algorithm 3. The level of Γ-radius sampling is a tuned value for all heuristics, and is also a maximum level of sampling when a stopping condition is employed by the heuristic.

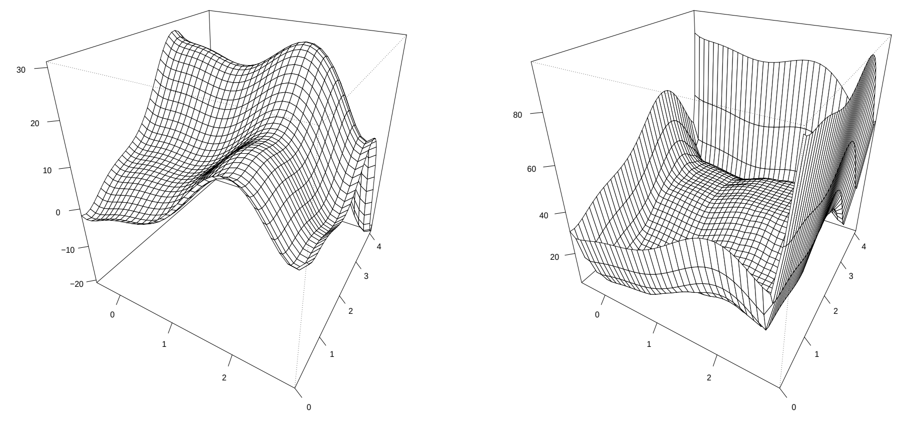
Each run of each heuristic identifies an estimate of the location and value of a robust global optimum. The robust global value used by the heuristic is likely to be an inaccurate estimate of the actual worst case value at the identified location. Therefore we post process all estimated values by randomly sampling 1,000,000 points in the Γ-uncertainty neighbourhood of the identified robust location, and taking the maximum sampled value as the estimated worst case cost. This robust value is taken as the output of a single heuristic-test problem run. This post processing does not impact on the heuristic search.

We employ ten multi-dimensional test functions over six dimensions: 2D, 5D, 10D, 30D, 60D and 100D, plus the 2D polynomial test problem from [BNT10b]. This gives 61 test problem instances for each of the heuristics to be applied to. Each heuristic is applied to each test instance 200 times to give reasonable sample sets of results for comparison, in order to identify the best performing approach. In line with assumed restrictions on numbers of function evaluations when handling real-world problems, each test problem run is limited to a budget of 5,000 function evaluations. Prior to undertaking the sample runs parameter tuning has been applied, as described in Section 3.6.2.

Name	$\mathcal{X}$	Γ
Rastrigin	$[14.88, 25.12]^n$	0.5
MultipeakF1	$[-5, -4]^n$	0.0625
MultipeakF2	$[10, 20]^n$	0.5
Branke's Multipeak	$[-7, -3]^n$	0.5
Pickelhaube	$[-40, -20]^n$	1
Heaviside Sphere	$[-30, -10]^n$	1
Sawtooth	$[-6, -4]^n$	0.2
Ackley	$[17.232, 82.768]^n$	3
Sphere	$[15, 25]^n$	1
Rosenbrock	$[7.952, 12.048]^n$	0.25
2D polynomial	$[-1, 4]^2$	0.5

Table 3.1: Test functions.

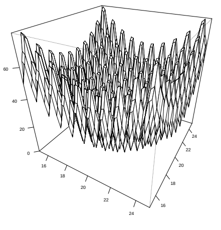
Table 3.1 presents the eleven test functions used within our experimental testing. The functions are based on the literature: [Bra98, KEB10, KRD+11, Kru12, JY13, BNT10b], and their mathematical description is given in Appendix 3.8.2. 3D plots of the 2D versions of these multi-dimensional functions are shown in Figures 3.4 and 3.5.

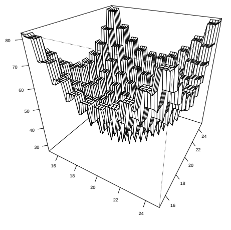


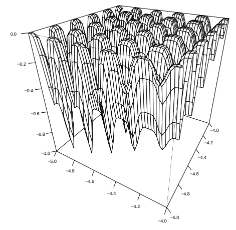
(a) 2D Polynomial Nominal (b) 2D Polynomial Worst

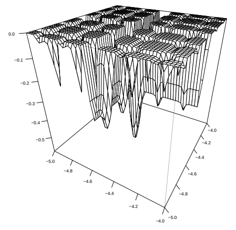
Figure 3.4: Plots of 2D Polynomial test function [BNT10b] in the rPSO test suite.

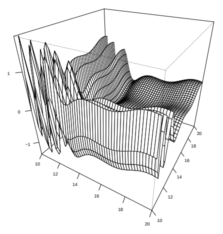
The common features of the objective function surfaces of individual test problems means that such problems can be associated with high level categorisations such as

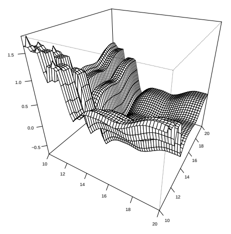

(a) Rastrigin Nom

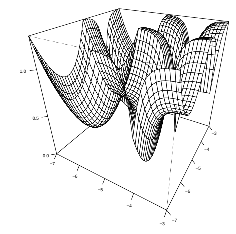

(b) Rastrigin Worst

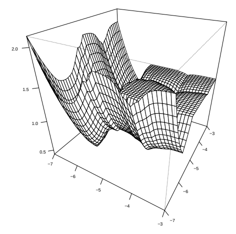

(c) Multipeak F1 Nom

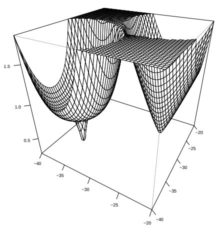

(d) Multipeak F1 Worst

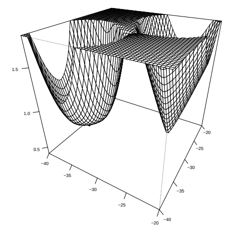

(e) Multipeak F2 Nom

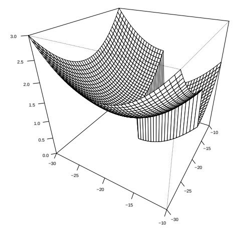

(f) Multipeak F2 Worst

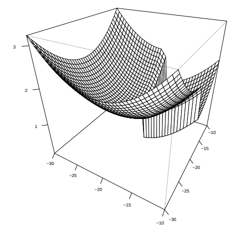

(g) Brankes Multi Nom

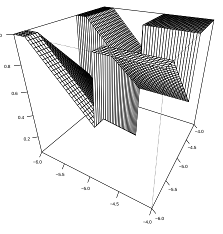

(h) Brankes Multi Worst

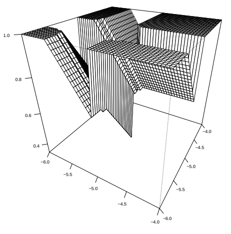

(i) Pickelhaube Nom

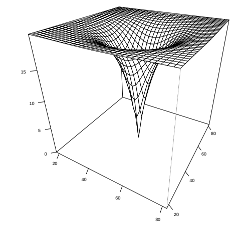

(j) Pickelhaube Worst

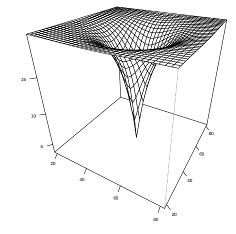

(k) Heaviside S Nom

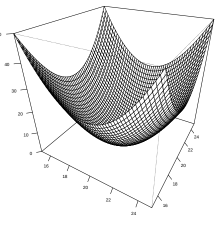

(l) Heaviside S Worst

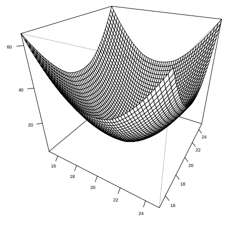

(m) Sawtooth Nom

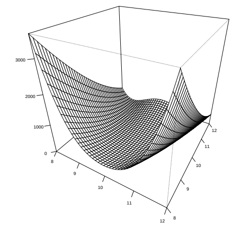

(n) Sawtooth Worst


(o) Ackley Nom


(p) Ackley Worst


(q) Sphere Nom


(r) Sphere Worst


(s) Rosenbrock Nom

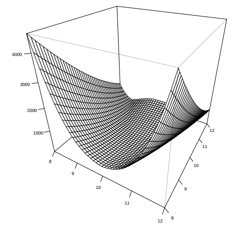

(t) Rosenbrock Worst

Figure 3.5: Plots of 2D versions of the 10 multi-dimensional problems in the rPSO test suite.

multi-modality, basins or valleys, see e.g. [JY13]. Here we have arranged the ten multi-dimensional test problems in an approximate order based on modality and common features, which should be apparent in the 3D plots shown in Figure 3.5.

All of our algorithms have been coded in Java, with calls to the mathematical programming software IBM ILOG CPLEX Optimization Studio V12.6.3 to solve the second order cone problem element of a d.d. calculation.

3.6.2 Parameter tuning

Parameter tuning has been undertaken at the dimensional level, separately for each heuristic, and for each of the six dimensions. This seems reasonable in a practical setting, where a decision maker is likely to have a good advance understanding of the dimension of the problem at hand. This generates a set of parameters for each heuristic separately for each dimension in the testing process.

Our tuning uses an evolutionary tuning approach applied to a subset of the test suite (four instances), including only problems where the nominal and robust global optima are differently located. This is primarily to ensure that the tuned level of inner maximisation is not biased by test problems where the nominal and robust optima are coincident, and therefore where no Γ-uncertainty neighbourhood analysis might be desirable.

Within the tuning GA each member of the evolving population represents a set of parameter values for a given heuristic operating at a specific dimension. For all heuristics the same level of tuning is employed for a given dimension: each problem in the tuning subset of the test suite is run for the same number of samples, with the same population size and number of generations used. For a given member of the evolutionary population the mean of the sample runs on a single tuning test problem is calculated, and ranked in comparison to the other individuals in the population. The average ranking of these means over the tuning test suite is used as the measure of utility within an evolutionary tournament selection, see e.g. [ES12].

For all heuristics a key tuned parameter is the extent of the inner maximisation analysis. For d.d. the parameters that control the parameter σ which impacts the determination of high cost points, and the minimum step size required for any descent direction step, are tuned. This also applies to our framework when set to employ a d.d. calculation. For LEH the parameters that control the genetic algorithm employed in the identification of the largest empty hypersphere devoid of all high cost points, see equation (lehMM), are tuned. Again this applies to our framework when set to employ an LEH calculation. For all four rPSO based heuristics (baseline rPSO and the three alternative framework settings) the new parameters that are tuned are stated in Sections 3.3.4, 3.5.2, 3.5.3 and 3.5.4. Tuned parameter values are given in Appendix 3.8.4.

3.6.3 Results

Here we present results of the 200 sample runs for each of the six robust heuristics when applied to each of the 61 test problem-dimension instances. The mean results are shown in Tables 3.2 and 3.3. The best or statistically equivalent to the best results (best-equivalent) due to Wilcoxon rank-sum testing with 95% confidence and employing a Bonferonni correction (see e.g. [HTF09]) are highlighted. That is a highlight on a method means that no other method is statistically better. For Table 3.3 this applies at the cell level. Note that being best with respect to the mean objective value does not always correspond to being statistically best-equivalent. Statistical analysis was undertaken in R, see [R C19, Din17]. Details of the distributions of the 200 samples runs for the multi-dimensional problems are provided by box plots in Appendix 3.8.3. The box plots for the 2D Bertsimas polynomial are shown in Figure 3.6.

Heuristic	Mean
d.d.	6.91
LEH	**4.80**
rPSO (4)	6.10
rPSOdd $(4,5)$	5.97
rPSOleh $(4,6)$	7.13
rPSOlehdd $(4,5,6)$	5.29

Table 3.2: Mean results for 200 sample runs for the 2D polynomial function due to [BNT10b]. Statistically equivalent best heuristics are highlighted. Bracketed numbering on rPSO based heuristics refers to the outer minimisation algorithms used.

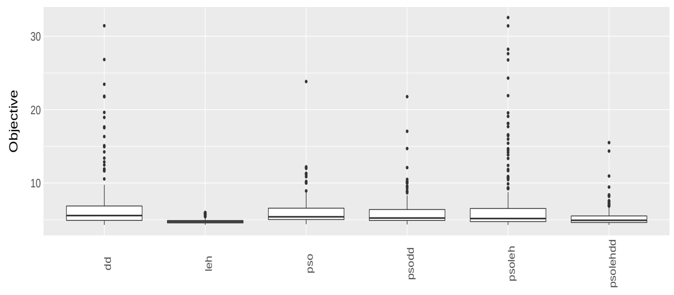


Figure 3.6: Box plots of 2D Polynomial test function [BNT10b] robust objective values for 200 sample runs.

		Rastrigin	MultipeakF1	MultipeakF2	Brankes	Pickelhaube	Heaviside	Sawtooth	Ackley	Sphere	Rosenbrock
2D	d.d.	37.15	-0.57	-0.62	0.44	0.40	0.58	0.59	11.00	1.06	9.37
	LEH	34.56	-0.60	-0.67	0.44	0.40	0.33	0.47	9.50	1.11	7.59
	rPSO (2)	39.23	-0.47	-0.54	0.48	0.52	0.30	0.51	11.58	1.56	11.36
	rPSOdd (2,3)	38.42	-0.51	-0.56	0.48	0.51	0.57	0.53	9.89	1.13	9.13
	rPSOleh (2,4)	37.42	-0.61	-0.68	0.43	0.38	1.01	0.63	9.38	1.01	7.71
	rPSOlehdd (2,3,4)	36.22	-0.61	-0.65	0.43	0.39	0.74	0.56	9.37	1.01	7.72
5D	d.d.	67.72	-0.44	-0.41	0.45	0.33	1.04	0.53	12.43	1.04	13.32
	LEH	76.18	-0.47	-0.55	0.42	0.54	0.76	0.47	13.01	2.81	38.73
	rPSO (2)	77.68	-0.48	-0.54	0.47	0.68	0.97	0.47	11.45	2.83	40.98
	rPSOdd (2,3)	73.85	-0.50	-0.60	0.41	0.54	1.05	0.52	8.49	1.61	24.49
	rPSOleh (2,4)	65.30	-0.60	-0.69	0.37	0.44	1.06	0.48	7.59	1.08	19.57
	rPSOlehdd (2,3,4)	67.50	-0.53	-0.56	0.41	0.48	1.02	0.44	7.86	1.36	20.18
10D	d.d.	108.47	-0.46	-0.25	0.56	0.90	1.83	0.59	19.45	1.30	30.77
	LEH	128.19	-0.44	-0.51	0.65	0.93	1.29	0.49	15.16	10.22	135.68
	rPSO (2)	123.10	-0.47	-0.51	0.57	0.81	1.16	0.48	13.00	6.97	111.34
	rPSOdd (2,3)	137.53	-0.44	-0.51	0.51	0.47	1.03	0.46	7.56	2.37	29.52
	rPSOleh (2,4)	101.89	-0.52	-0.60	0.48	0.50	1.04	0.43	8.19	2.55	51.19
	rPSOlehdd (2,3,4)	116.21	-0.55	-0.65	0.40	0.44	1.02	0.43	7.18	2.18	28.70
30D	d.d.	323.45	-0.44	-0.33	0.70	1.75	7.50	0.66	21.42	88.11	3951.97
	LEH	334.08	-0.44	-0.46	0.70	1.40	1.93	0.49	15.54	39.70	556.96
	rPSO (2)	379.59	-0.43	-0.28	0.63	1.07	2.20	0.50	16.62	37.54	1149.84
	rPSOdd (2,3)	360.35	-0.45	-0.29	0.62	0.87	1.31	0.60	20.72	7.91	241.75
	rPSOleh (2,4)	226.57	-0.63	-0.51	0.47	0.44	1.06	0.35	6.78	5.30	104.00
	rPSOlehdd (2,3,4)	310.65	-0.45	-0.40	0.54	1.64	1.03	0.44	18.62	2.86	89.24
60D	d.d.	876.26	-0.35	-0.23	0.75	1.81	15.72	0.67	21.26	365.66	17197.36
	LEH	671.26	-0.44	-0.42	0.70	1.60	4.21	0.45	17.50	85.63	2111.79
	rPSO (2)	693.70	-0.44	-0.41	0.67	1.79	5.43	0.49	18.59	119.83	3186.00
	rPSOdd (2,3)	422.03	-0.60	-0.47	0.51	0.66	1.03	0.50	20.34	2.84	108.69
	rPSOleh (2,4)	553.54	-0.52	-0.47	0.61	0.74	1.41	0.37	10.49	20.03	323.83
	rPSOlehdd (2,3,4)	503.94	-0.55	-0.49	0.58	1.12	2.45	0.37	14.66	41.93	806.49
100D	d.d.	1677.52	-0.33	-0.23	0.77	1.81	27.79	0.69	21.29	711.25	37264.55
	LEH	975.05	-0.50	-0.43	0.78	1.63	3.32	0.33	14.46	61.66	1302.55
	rPSO (2)	1177.93	-0.43	-0.40	0.69	1.81	9.06	0.49	18.77	207.99	5968.79
	rPSOdd (2,3)	648.67	-0.58	-0.49	0.56	1.77	3.55	0.42	17.65	36.11	1288.00
	rPSOleh (2,4)	842.71	-0.53	-0.48	0.62	1.76	5.45	0.40	16.67	119.76	2975.69
	rPSOlehdd (2,3,4)	720.08	-0.48	-0.45	0.70	1.72	4.55	0.41	17.22	82.90	2039.80

Table 3.3: Mean results for 200 sample runs for the 10 multi-dimensional problems. Statistically equivalent best heuristics are highlighted. Bracketed numbering on rPSO based heuristics refers to the outer minimisation algorithms used.

We see that the 2D instances are dominated by rPSOleh, baseline LEH and rPSOlehdd, which are best-equivalent in 64%, 45% and 36% of instances respectively; here, e.g. 64% refers to rPSOleh being best or statistically equivalent to the best in seven out of the eleven 2D test problem instances. For 5D, rPSOleh is best-equivalent in 60% of cases, followed by the baseline d.d. with 50%, rPSOlehdd 30%, and the baseline LEH 10%. However for 10D the picture changes, with rPSOlehdd dominating as best-equivalent in 70% of cases, followed by rPSOleh and the baseline d.d. with 20% each.

The 30D and 60D instances are completely dominated by the new framework. For 30D rPSOleh is best-equivalent in 90% of cases, with only rPSOlehdd achieving any other best-equivalent results, in 30% of cases. At 60D rPSOdd comes into prominence, being best-equivalent in 70% of cases, with 30% for rPSOleh and 20% for rPSOlehdd. Finally for 100D rPSOdd is again best-equivalent in 70% of cases, although now the baseline LEH heuristic is best-equivalent in 50% of cases, with 10% for rPSOleh.

Table 3.4 summarises the proportion of the 61 test problem instances for which each heuristic is identified as the best or statistically equivalent to the best. The three settings in the new framework lead the order of best to worst results: rPSOleh, rPSOlehdd, rPSOdd, LEH, and d.d., with rPSO failing to be the best for any test instance.

Heuristic	Best-equiv.
d.d.	11.48%
LEH	18.03%
rPSO (4)	00.00%
rPSOdd (4,5)	22.95%
rPSOleh (4,6)	44.26%
rPSOlehdd (4,5,6)	31.15%

Table 3.4: Summary of the proportion of best or statistically equivalent to the best results for each heuristic. Bracketed numbering on rPSO based heuristics refers to the outer minimisation algorithms used.

From the summary Table 3.4 it is not clear how each of the settings for the new rPSO framework perform individually against the three baseline comparators. Therefore we have conducted a number of further statistical tests. First we compared rPSOdd against the three baselines, rPSOleh against the baselines, and rPSOlehdd against the baselines. The result is that rPSOdd is best-equivalent in 49.2% of the 61 test instances, rPSOleh is best-equivalent in 75.4% of instances, and rPSOlehdd is best-equivalent in 65.6% of instances. In each test the baseline LEH heuristic had the next best performance, with best-equivalent results of 39.4%, 21.3% and 29.5% of the test instances respectively.

Finally we compared each of the settings for the new rPSO framework individually against each of the three baseline comparators, in a series of one-to-one analyses. The results are summarised in Table 3.5, where each cell comprises three values: top, middle and bottom. Each cell shows the proportion of the 61 test problem instances for which each new framework heuristic is better than (top), statistically equivalent to (middle), or worse than (bottom) each baseline heuristic. Within each set of three values the highest is highlighted.

In the comparison between the rPSO and rPSOdd methods, specifically considering the impact of augmenting the baseline rPSO velocity equations (Vel1) with a d.d. component (Vel2), rPSO is best or statistically equivalent to the best in 23.0% of the 61 test instances, with rPSOdd best-equivalent in 88.5%. The dominance of the new rPSO framework in one-to-one comparisons against each of the baselines is clear, in particular for rPSOleh and rPSOlehdd. Indeed rPSOleh is individually better than each of the baselines in nearly 80% of test instances.

	d.d.	LEH	rPSO (4)
	65.6%	**52.5%**	**77.0%**
rPSOdd (4, 5)	6.6%	8.2%	11.5%
	27.9%	39.3%	11.5%
	86.9%	**78.7%**	**93.4%**
rPSOleh (4, 6)	3.3%	3.3%	3.3%
	9.8%	18.0%	3.3%
	86.9%	**70.5%**	**91.8%**
rPSOlehdd (4, 5, 6)	3.3%	3.3%	1.6%
	9.8%	26.2%	6.6%

Table 3.5: Results of one-to-one statistical tests between the new framework and baseline heuristics. Each cell shows the percentage of test problem instances where: (top) the new heuristic is best, (middle) the new and baseline heuristics are equivalent, and (bottom) the baseline heuristic is best. Bracketed numbering on rPSO based heuristics refers to the outer minimisation algorithms used.

3.7 Conclusions and further work

We have developed a new robust metaheuristic framework for box-constrained, black-box robust optimisation problems under implementation uncertainty. Our robust approach assumes min max conditions, seeking to find solutions that optimise the worst performance. Our new approach uses a baseline robust particle swarm population based heuristic as a frame, adapting elements of two existing individual-based robust metaheuristics, descent directions [BNT10b] and largest empty hypersphere [HGW19], and combing them along with new features. The following novel features are introduced here:

- An extension of the PSO movement formulation to include particle level, iteration level, d.d. vector information: exploiting uncertainty neighbourhood information in order to provide a locally optimal directional movement component.

- Efficiency savings in terms of numbers of function evaluations due to the use of a particle level stopping condition.
- The introduction of the concept of dormancy, whereby the repeated non-requirement to perform any function evaluations is monitored at a particle level in order to interrupt particles trapped in previously visited regions or outside of the feasible region.
- The relocating of dormant particles by an optimal exploration-focussed calculation of the largest empty hypersphere devoid of all high cost points.

This results in a framework encompassing a full enhanced capability plus two settings where specific enhancements are 'switched off'. With the full capability (rPSOlehdd), the baseline rPSO heuristic is augmented with an additional component in the standard PSO velocity equation, using the descent direction vector that optimally points away from the worst Γ-uncertainty neighbourhood points around a candidate point (particle location). This is further augmented with both the stopping condition and the determination of the largest hypersphere empty of previously evaluated poor points, from the LEH heuristic. The stopping condition allows efficiencies in the inner maximisation calculations, which are terminated early if any Γ-neighbourhood point is identified with nominal function value worse than a particle's current personal best information. The calculation of an LEH is used to relocate particles that have become 'dormant', either due to repeated movements outside of the feasible region or repeated movements in areas of the feasible region where points with high nominal function value have already been identified.

In the alternative framework settings, the baseline rPSO heuristic can be augmented by our enhanced d.d. capability alone (rPSOdd), or by our enhanced LEH capability alone (rPSOleh).

The performance of the new framework has been assessed by applying it to 61 test problem instances, covering six dimensions up to 100D, a single 2D problem and 10 multi-dimensional problems. The performance of our framework has been compared against three existing baseline approaches, a repeating d.d. approach, LEH, and a baseline robust PSO. Our new framework is shown to outperform the existing approaches across all dimensions. For 10D, 30D and 60D instances the new framework dominates. It also outperforms the baseline heuristic for other dimensions, although both LEH and d.d. also produce some good results.

One potential extension of our new framework is to undertake explicit inner maximisation searches, for example using PSO or GA searches, as opposed to the use of uniform random sampling here. Also, in order to make the technique more widely applicable other forms of uncertainty, for example model uncertainty, could be accommodated.

The current focus on a Γ-radius uncertainty neighbourhood can also be extended, to the consideration of other descriptions of a point's uncertainty neighbourhood.

3.8 Appendices

3.8.1 List of Abbreviations

Abbreviation	Definition
d.d.	descent directions
GA	genetic algorithm
hcp	high cost point
LEH	largest empty hypersphere
PSO	particle swarm optimisation
rPSO	robust particle swarm optimisation
rPSOdd	robust particle swarm optimisation with descent directions
rPSOleh	robust particle swarm optimisation with largest empty hypersphere
rPSOlehdd	robust particle swarm optimisation with descent directions and largest empty hypersphere

Table 3.6: Commonly used abbreviations.

3.8.2 Test functions

Functions used in the experimental testing of the enhanced rPSO framework. These functions are based on [Bra98, KEB10, KRD+11, Kru12, JY13, BNT10b].

Rastrigin

$$f(\boldsymbol{x}) = 10n + \sum_{i=1}^{n} \left[(x_i - 20)^2 - 10\cos(2\pi(x_i - 20))\right]$$

The feasible region is the hypercube $x_i \in [14.88, 25.12]$.

MultipeakF1

$$f(\boldsymbol{x}) = -\frac{1}{n}\sum_{i=1}^{n} g(x_i)$$

$$g(x_i) = \begin{cases} e^{-2\ln 2(\frac{(x_i+5)-0.1}{0.8})^2}\sqrt{|\sin(5\pi(x_i+5))|} & \text{if } 0.4 < x_i + 5 \leq 0.6 \ , \\ e^{-2\ln 2(\frac{(x_i+5)-0.1}{0.8})^2}\sin^6(5\pi(x_i+5)) & \text{otherwise} \end{cases}$$

The feasible region is the hypercube $x_i \in$ [-5, -4].

MultipeakF2

$$f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} g(x_i)\ ,\ g(x_i) = 2\sin(10e^{-0.2(x_i-10)}(x_i-10))e^{-0.25(x_i-10)}$$

The feasible region is the hypercube $x_i \in$ [10, 20].

Branke's Multipeak

$$f(\boldsymbol{x}) = \max\{c_1, c_2\} - \frac{1}{n}\sum_{i-1}^{n} g(x_i)$$

$$g(x_i) = \begin{cases} c_1\left(1 - \frac{4((x_i+5)+\frac{b_1}{2})^2}{b_1^2}\right) & \text{if } -b_1 \le (x_i+5) < 0\ , \\ c_2 \cdot 16^{\frac{-2|b_2-2(x_i+5)|}{b_2}} & \text{if } 0 \le (x_i+5) \le b_2\ , \\ 0 & \text{otherwise} \end{cases}$$

Here $b_1 = 2$, $b_2 = 2$, $c_1 = 1$, $c_2 = 1.3$.
The feasible region is the hypercube $x_i \in$ [-7, -3].

Pickelhaube

$$f(\boldsymbol{x}) = \frac{5}{5-\sqrt{5}} - \max\{g_0(\boldsymbol{x}), g_{1a}(\boldsymbol{x}), g_{1b}(\boldsymbol{x}), g_2(\boldsymbol{x})\}$$

$$g_0(\boldsymbol{x}) = \frac{1}{10}e^{-\frac{1}{2}\|\boldsymbol{x}+30\|}$$

$$g_{1a}(\boldsymbol{x}) = \frac{5}{5-\sqrt{5}}\left(1 - \sqrt{\frac{\|\boldsymbol{x}+30+5\|}{5\sqrt{n}}}\right)$$

$$g_{1b}(\boldsymbol{x}) = c_1\left(1 - \left(\frac{\|\boldsymbol{x}+30+5\|}{5\sqrt{n}}\right)^4\right)$$

$$g_2(\boldsymbol{x}) = c_2\left(1 - \left(\frac{\|\boldsymbol{x}+30-5\|}{5\sqrt{n}}\right)^{d_2}\right)$$

Here $c_1 = 625/624$, $c_2 = 1.5975$, $d_2 = 2 = 1.1513$.
The feasible region is the hypercube $x_i \in$ [-40, -20].

Heaviside Sphere

$$f(\boldsymbol{x}) = \left(1 - \prod_{i=1}^{n} g(x_i)\right) + \sum_{i=1}^{n} \left(\frac{(x_i + 20)}{10}\right)^2$$

$$g(x_i) = \begin{cases} 0 & \text{if } 0 < (x_i + 20) \ , \\ 1 & \text{otherwise} \end{cases}$$

The feasible region is the hypercube $x_i \in$ [-30, -10].

Sawtooth

$$f(\boldsymbol{x}) = 1 - \frac{1}{n} \sum_{i=1}^{n} g(x_i) \ , \ g(x_i) = \begin{cases} (x_i + 5) + 0.8 & \text{if } -0.8 \leq (x_i + 5) < 0.2 \ , \\ 0 & \text{otherwise} \end{cases}$$

The feasible region is the hypercube $x_i \in$ [-6, -4].

Ackleys

$$f(\boldsymbol{x}) = -20 \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - 50)^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi(x_i - 50))\right) + 20 + \exp(1)$$

The feasible region is the hypercube $x_i \in$ [17.232, 82.768].

Sphere

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} (x_i - 20)^2$$

The feasible region is the hypercube $x_i \in$ [15, 25].

Rosenbrock

$$f(\boldsymbol{x}) = \sum_{i=1}^{n-1} [100((x_{i+1} - 10) - (x_i - 10)^2)^2 + ((x_i - 10) - 1)^2]$$

The feasible region is the hypercube $x_i \in$ [7.952, 12.048].

2D polynomial

$$f(x) = 2x_1^6 - 12.2x_1^5 + 21.2x_1^4 + 6.2x_1 - 6.4x_1^3 - 4.7x_1^2 - x_2^6 - 11x_2^5 + 43.3x_2^4$$
$$-10x_2 - 74.8x_2^3 + 56.9x_2^2 - 4.1x_1x_2 - 0.1x_2^2x_1^2 + 0.4x_2^2x_1 + 0.4x_1^2x_2$$

The feasible region is the square $x_i \in$ [-1, 4].

3.8.3 Box plots

Box plots of the results of the experimental testing on our comparator robust heuristics applied to 60 test problems covering the 10 multi-dimensional problems. Each plot is based on 200 sample runs of each heuristic applied to each problem instance.

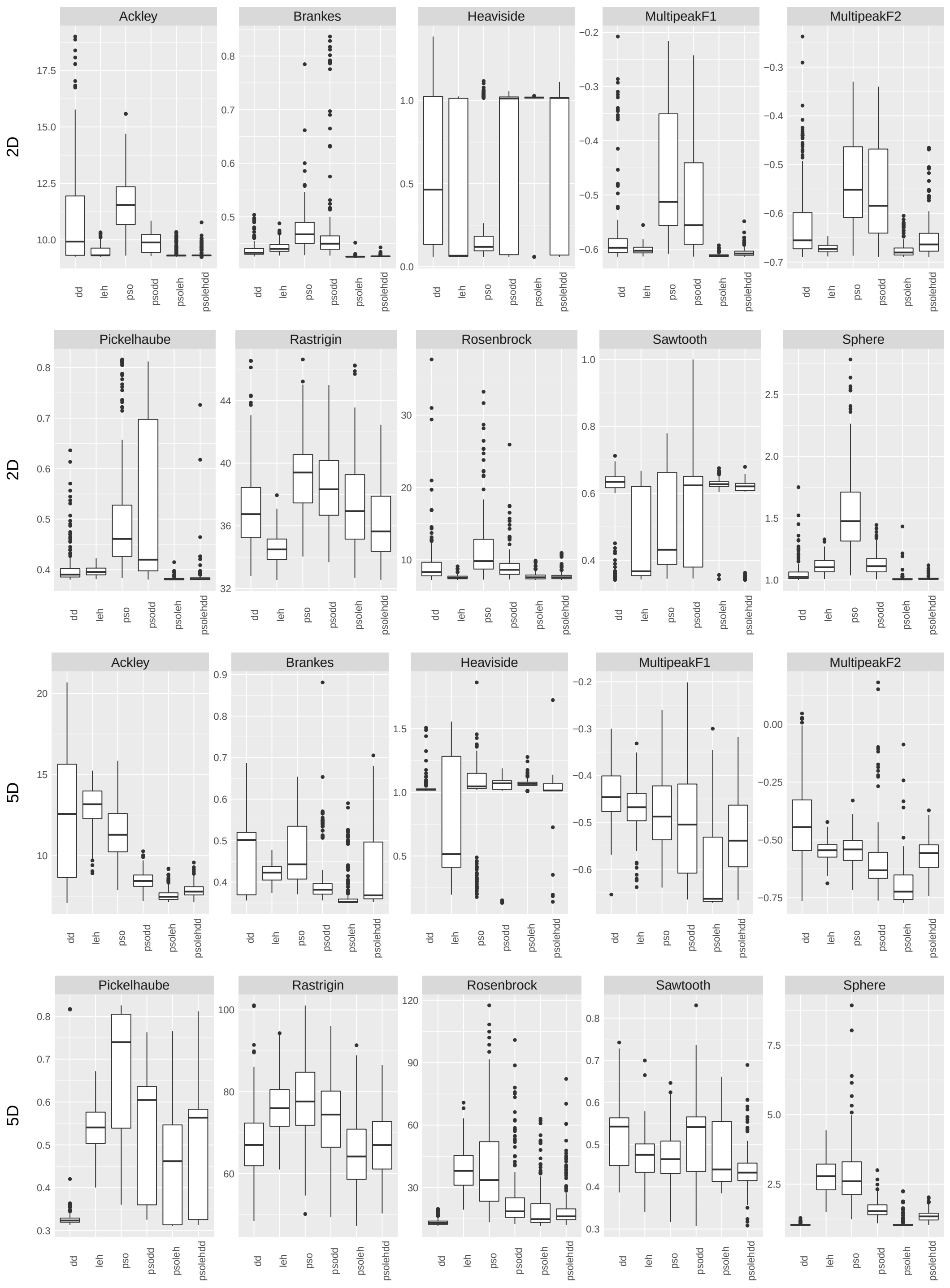


Figure 3.7: Box plots of 2D and 5D robust objective values for 200 sample runs.

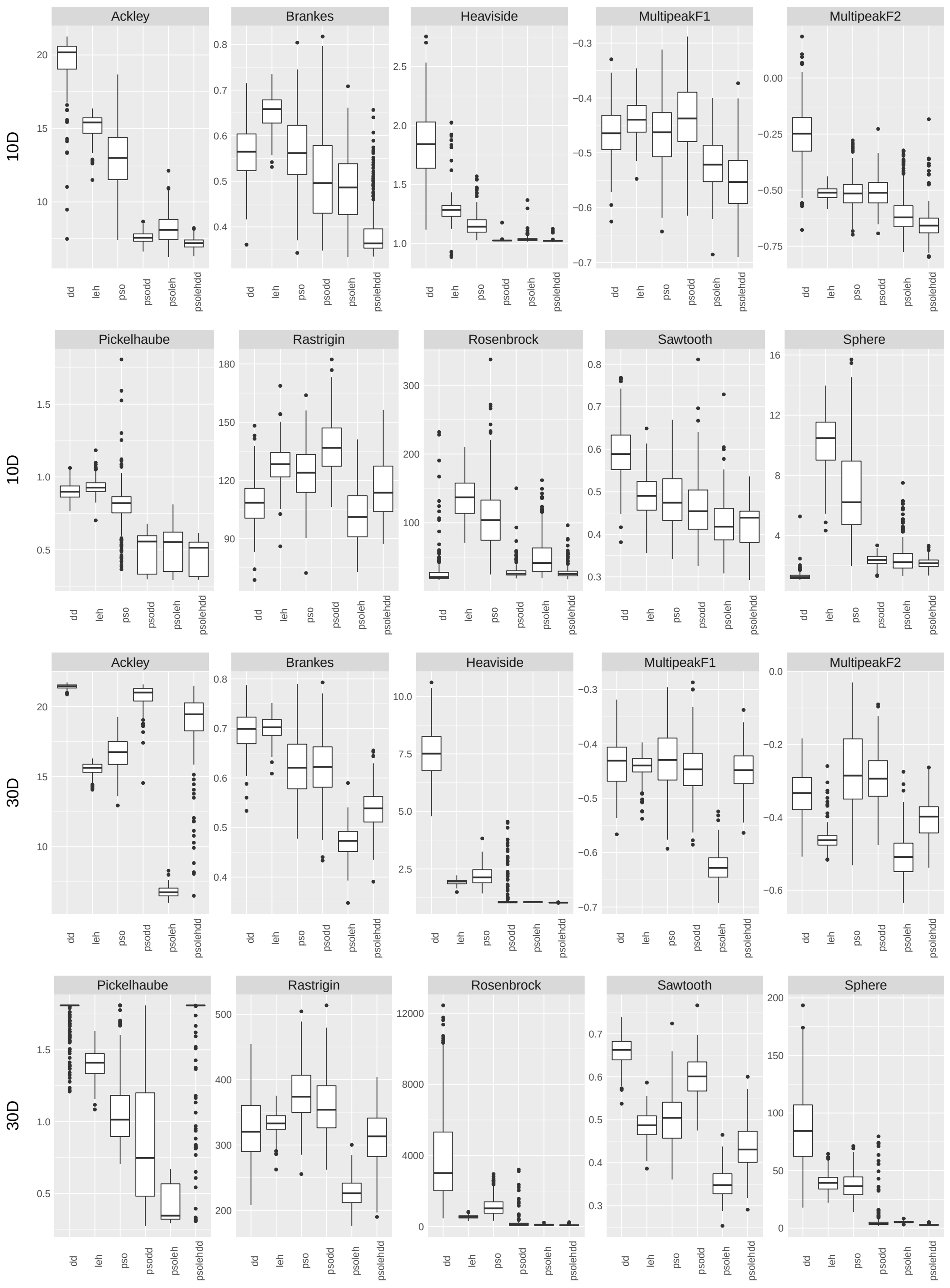


Figure 3.8: Box plots of 10D and 30D robust objective values for 200 sample runs.

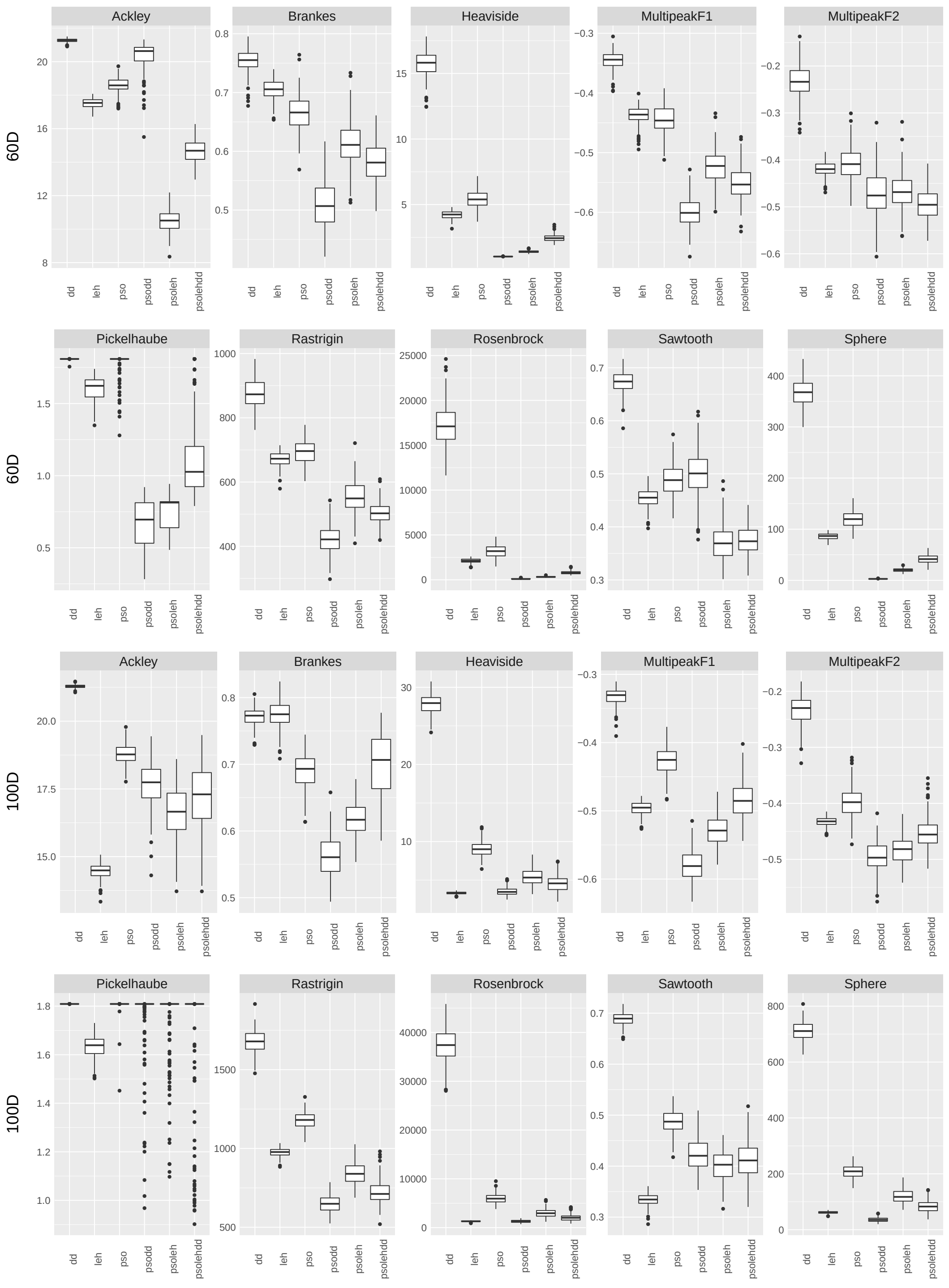


Figure 3.9: Box plots of 60D and 100D robust objective values for 200 sample runs.

3.8.4 Heuristic parameter values

As described in Section 3.6.2 the parameter values for all comparator heuristics have been tuned at the dimensional level – generating a set of dimension-specific parameters for each heuristic. These values were then used in the generation of the experimental results given in Section 3.6.3. The values of those tuned parameters are given below in Tables 3.7 to 3.12. For all heuristics the extent of the inner maximisation search is a tuned parameter.

For the d.d. heuristic 5 parameters were tuned in addition to the extent of the inner maximisation: σ_{init} the initialisation factor for the high cost set threshold $\sigma(t)$ at step t, the $\sigma(t)$ reduction factor α used to adapt the threshold value, σ_α a lower threshold for $\sigma(t)$, a factor $init\rho_{Min}$ which is multiplied by the uncertainty parameter Γ in order to establish a minimum step size for the search, and ρ_{red} a reduction factor for reducing this minimum step size with every step t. See [BNT10b] for a full description of these parameters. The tuned values for these parameters are given in Table 3.7.

		σ_α	α	σ_{init}	ρ_{red}	$init\rho_{Min}$	inner
d.d.	2D	0.0057	1.0648	0.3856	0.9252	0.0238	15
	5D	0.0067	1.0108	0.3548	0.9228	0.0730	18
	10D	0.0090	1.0127	0.2135	0.9671	0.0145	5
	30D	0.0043	1.0204	0.2651	0.9343	0.0978	19
	60D	0.0047	1.0409	0.1022	0.9689	0.0478	5
	100D	0.0080	1.0634	0.1482	0.9493	0.0120	8

Table 3.7: Tuned parameter values for the d.d. heuristic.

For the LEH heuristic which employs a genetic algorithm to search for the centre of the largest empty hypersphere, 6 parameters associated with the GA were tuned in addition to the extent of the inner maximisation: the size of the population, number of generations, number of elites, tournament size, and mutation probability and size. Here mutation 'size' is actually a percentage value which is subsequently multiplied by the dimensional range of the decision variable space $\mathcal{X}$ in order to specify the actual amount by which any value is adjusted due to mutation. The tuned values for these parameters are given in Table 3.8.

For the baseline rPSO heuristic 4 parameters were tuned in addition to the extent of the inner maximisation: the C_1 and C_2 acceleration parameters, the inertia weight parameter ω, and the swarm size. See [SE98] for a further description of these parameters. The tuned values for these parameters are given in Table 3.9. These parameters are also tuned for all rPSO based heuristics. It should be noted that there is no need to tune the number of iterations as this is controlled by the budget limitation on the number of function evaluations.

		pop	gens	elites	tour size	mut prob	mut size	inner
LEH	2D	20	5	5	19	0.8000	0.2000	249
	5D	4	25	2	2	0.2480	0.2178	42
	10D	5	20	2	2	0.0000	0.0800	242
	30D	10	10	1	7	0.3400	0.0800	16
	60D	5	20	0	3	0.0120	0.0400	84
	100D	20	5	0	10	0.5500	0.0000	114

Table 3.8: Tuned parameter values for the LEH heuristic.

		C_1	C_2	ω	swarm	inner
rPSO	2D	0.1184	1.7000	0.7056	43	45
	5D	1.0212	1.0816	0.3528	34	28
	10D	0.9204	0.8000	0.5000	15	23
	30D	2.1800	2.0424	0.3800	283	7
	60D	0.6300	1.1248	0.1100	49	16
	100D	0.4400	1.2200	0.5880	21	22

Table 3.9: Tuned parameter values for the baseline rPSO heuristic.

For the rPSOdd heuristic in addition to the baseline rPSO parameters and the extent of the inner maximisation, 4 other parameters were tuned: the new C_3 parameter and 3 of the d.d. parameters: the initial σ factor for the high cost set threshold, σ_{limit} the lower threshold for σ, and the factor $init\rho_{Min}$ which is multiplied by the uncertainty parameter Γ in order to establish a minimum step size for the search. Note that unlike in the local individual based d.d. approach, in the modified rPSO formulation the initial σ, the σ_{limit} and the $init\rho_{Min}$ values are re-employed separately for each particle and at each iteration of the swarm. The factor α used to adapt the threshold value is not used as in order to limit processing time for the experiments σ is reduced a fixed ten times in ten equal steps from the initial σ value to the σ_{limit} value. Furthermore the ρ_{red} reduction factor employed in d.d. is not used, as for each particle at each iteration the magnitude of a calculated descent direction vector is checked solely against the minimum step size based on $init\rho_{Min}$. These 4 parameters are also tuned for the rPSOlehdd heuristic. The tuned values for these parameters are given in Table 3.10.

For the rPSOleh heuristic in addition to the baseline rPSO parameters and the extent of the inner maximisation, 7 other parameters were tuned: the 6 parameters associated with the LEH genetic algorithm were tuned (size of the population, number of generations, number of elites, tournament size, and mutation probability and size), plus $dormancy_{limit}$ the number of iterations over which a particle must be dormant prior to being moved using the LEH calculation. These 7 parameters are also tuned for the rPSOlehdd heuristic. The tuned values for these parameters are given in Table 3.11.

For the rPSOlehdd heuristic in addition to the baseline rPSO parameters and the extent of the inner maximisation, the 4 additional parameters associated with rPSOdd and 7 additional parameters associated with rPSOleh were tuned. The tuned values for these parameters are given in Table 3.12.

		C_1	C_2	ω	swarm	inner	C_3	σ_{lim}	σ	$init\rho_{Min}$
rPSOdd	2D	2.3573	0.3273	0.4082	11	59	3.5742	0.0064	0.2513	0.029
	5D	0.2829	2.3152	0.1892	43	11	5.5863	0.002	0.1942	0.0272
	10D	2.3136	0.3113	0.2990	4	53	9.5752	0.0088	0.3145	0.0453
	30D	1.2549	2.0473	0.7530	9	47	3.9549	0.0078	0.2722	0.055
	60D	0.3714	0.3057	0.6141	2	18	6.0067	0.0074	0.3956	0.0969
	100D	0.5788	0.5688	0.7671	9	10	8.6358	0.0015	0.2792	0.0767

Table 3.10: Tuned parameter values for the rPSOdd heuristic.

		C_1	C_2	ω	swarm	pop	gens	elites	tour sz	mut prob	mut sz	$dorm_{lim}$	inner
rPSOleh	2D	1.8900	2.3100	0.4700	13	10	10	9	8	0.1800	0.3800	10	31
	5D	0.8800	1.0900	0.7300	8	10	10	9	9	0.6900	0.1600	9	12
	10D	0.5300	1.4300	0.7300	15	10	10	4	7	0.2900	0.3100	6	16
	30D	2.3900	0.8500	0.7000	3	4	25	1	3	0.6600	0.3300	7	2
	60D	0.4000	1.4500	0.5200	2	4	25	3	3	0.6500	0.0900	10	3
	100D	1.6000	0.7400	0.7400	8	25	4	3	3	0.2300	0.3500	7	9

Table 3.11: Tuned parameter values for the rPSOleh heuristic.

		C_1	C_2	ω	swarm	pop	gens	elites	tour sz	mut prob	mut sz	$dorm_{lim}$	inner	C_3	σ_{lim}	σ	$init\rho_{Min}$
rPSOlehdd	2D	1.3456	0.4535	0.4580	6	4	25	2	3	0.0923	0.2608	2	55	0.3447	0.0031	0.3544	0.0866
	5D	1.1483	1.7084	0.1012	12	10	10	4	4	0.1854	0.3587	2	41	2.8921	0.0041	0.3203	0.0080
	10D	1.0528	0.5479	0.6569	2	5	20	1	4	0.4954	0.0595	8	10	6.1057	0.0045	0.2903	0.0769
	30D	0.0628	1.7129	0.6011	2	5	20	1	4	0.3604	0.1269	6	48	6.5166	0.0077	0.1254	0.0775
	60D	0.6624	1.4989	0.7526	21	5	20	0	4	0.1879	0.2745	10	9	0.8607	0.0032	0.2696	0.0292
	100D	0.8688	0.9148	0.8136	5	20	5	1	2	0.8881	0.0634	9	19	8.6079	0.0058	0.2635	0.0555

Table 3.12: Tuned parameter values for the rPSOlehdd heuristic.

Chapter 4

Paper 3: Automatic Generation of Algorithms for Robust Optimisation Problems using Grammar-Guided Genetic Programming

Submitted to *Applied Soft Computing* in April 2020 [HGD20a].
Partially funded through EPSRC grants EP/L504804/1 and EP/M506369/1.

Author 1: Martin Hughes, Lancaster University, United Kingdom.
Author 2: Marc Goerigk, University of Siegen, Germany.
Author 3: Trivikram Dokka, Lancaster University, United Kingdom.

Abstract: We develop algorithms capable of tackling robust black-box optimisation problems, where the number of model runs is limited. When a desired solution cannot be implemented exactly the aim is to find a robust one, where the worst case in an uncertainty neighbourhood around a solution still performs well. To investigate improved methods we employ an automatic generation of algorithms approach: Grammar-Guided Genetic Programming. We develop algorithmic building blocks in a Particle Swarm Optimisation framework, define the rules for constructing heuristics from these components, and evolve populations of search algorithms for robust problems. Our algorithmic building blocks combine elements of existing techniques and new features, resulting in the investigation of a novel heuristic solution space. We obtain algorithms which improve upon the current state of the art. We also analyse the component level breakdowns of the populations of algorithms developed against their performance, to identify high-performing heuristic components for robust problems.

4.1 Introduction

The use of optimisation search techniques to investigate a decision variable solution space and identify good solutions is common when using models to support informed decision making. However the search may be impacted by issues such as model run times, the size of the solution space, and uncertainty, see [BTEGN09, GS16]. In this work we are concerned with optimisation under implementation uncertainty, and where some budget on the number of model runs restricts the search.

If a model can take the form of a mathematical program, optimisation may be tackled efficiently and exactly. Here we assume this is not the case, and instead some model is employed which from an optimisation perspective can be considered a black-box where decision variable values are input and an objective extracted. In this case only an approximate global optimum is sought, and so in this work we consider metaheuristic techniques applicable to general, likely non-convex problems.

With implementation uncertainty an ideal solution cannot be achieved exactly, so solutions are sought where all points in the uncertainty neighbourhood around a candidate still perform well. Such a situation is common in many real-world applications. For example in engineering, manufacturing or construction it may not be possible to meet design specifications exactly, in which case a solution is sought which is tolerant of some variation in the design variables. In scheduling, timetabling, logistics, queuing or supply chain management it is desirable to be able to tolerate some deviation in resource levels and transportation or processing times. See, for example, [PBJ06, BNT07, BNT10b, Kru12, GMT14, GS16].

When it is known how the uncertainty is distributed, the problem is one of stochastic optimisation, see [PBJ06, HdMB14]. Instead we assume the uncertainty takes the form of some set containing all uncertainty scenarios such as an interval, making the problem one of robust optimisation. Specifically a classic robust setting is considered, where the worst (inner maximum) model value in the uncertainty region around a candidate solution is sought in the context of an overarching (outer) minimisation objective, [BTN98]. We seek improved methods for robust general, black-box problems that can be employed in this setting. This necessitates a metaheuristic approach, as the requirement is beyond the scope of robust mathematical programming methods. The simplistic alternative of ignoring the uncertainty and using a classic optimisation method can produce sub-optimal results, see [BTEGN09, GS16].

The application of metaheuristics to real-world problems is complicated by the need to both identify an appropriate search technique and define the associated parameter settings. Both of these choices can significantly impact on the effectiveness of any search. These difficulties can be overcome through the automatic generation of metaheuristics, actively seeking good heuristics and avoiding the need for the manual determination of the search algorithm and parameter values, see e.g. [BGH^{+}13, POH^{+}14]. This is the approach adopted here. Such an approach has been applied to a number of practical problems, for example computing [SAMO03], machine scheduling [JJB07], timetabling [MPF09], the design of data mining algorithms [PF10], and agent-based transportation simulation [vLHVB$^+$12].

For a general problem setting where there is no knowledge of the nature of a model's objective function surface we seek to develop improved general search techniques for robust problems. This is targeted at a situation where the problem dimension is known and there is an appreciation of model run time limitations leading to some budget on model runs, both of which are reasonable assumptions for real-world problems. In an alternative setting there may be access to some previous model run information due to historic analysis, or approximate model run data, e.g. due to the availability of a complementary model with much reduced run times. In which case given some appreciation of the nature of a model's objective function surface an even more targeted technique may be automatically generated. Both general and more targeted settings are considered in our work.

To automatically generate robust metaheuristics here, a hyper-heuristic approach is employed, genetic programming (GP) [Noh11]. This is an evolutionary process where each individual in a population is an algorithm – here a metaheuristic for a robust problem. From the initial population some measure of fitness is determined for each heuristic, and a new generation established through typical evolutionary selection, combination and mutation processes. After multiple generations the fittest heuristic is chosen and applied to the problem at hand.

To facilitate the GP search, heuristic sub-components are generated. When combined correctly these algorithmic building blocks form a complete heuristic. The sub-components form a language, and the design rules specifying how they combine to create a heuristic represent a grammar. This is Grammar-Guided Genetic Programming (GGGP) [Noh11].

As with any evolutionary approach, GP employs combination and mutation operations to generate improved (fitter) solutions. However integrating sub-algorithms (computer sub-programs in the more general GP sense), may not be straightforward when the intention is to form a coherent, executable higher level algorithm. A common GGGP approach uses a tree-based representations of the overarching algorithm [MLIDLS14, CBP15, MP16, MP17]. This approach-representation is adopted here, where we specify heuristic sub-components in terms of a context-free grammar (CFG) and use standard tree-based random combination and mutation operators [MP16].

4.1.1 Contributions and outline

Improved global metaheuristics are developed for robust black-box problems under implementation uncertainty, for problems of 30 dimensions (30D) and 100D and assuming a budget of 2,000 model runs. A GGGP search of the solution space of heuristics for robust problems is used to identify the best approaches. The previously uninvestigated heuristic solution space comprises algorithmic building blocks that combine to form a complete particle swarm based heuristic. A large number of sub-components are developed using existing approaches and novel implementations.

New algorithms are tested on a suite of problems, and improved heuristics for general robust problems are identified. The significance of individual algorithmic sub-components is also assessed against heuristic performance. The effectiveness of an inner maximisation by random sampling on a small number of points and using a particle level stopping condition, is established. For the outer minimisation a small swarm of particles performs well, as does communication via a Global typology. The preferred particle movement uses an inertia based velocity equation plus specialised capabilities drawn from the largest empty hypersphere [HGW19, HGD20b] and descent directions [BNT07, BNT10a, BNT10b, HGD20b] heuristics.

In Section 4.2 we outline the optimisation problem of concern here, and current approaches for addressing it. We include descriptions of the heuristics that form the basis for the building blocks in the GP analysis. Section 4.3 gives an overview on the automatic generation of algorithms, and in Section 4.4 GP is discussed in detail. This includes sub-component descriptions, the design rules for constructing complete heuristics, and our GP approach including tree-based representation and operators. Section 4.5 describes the experimental analysis, results for the best heuristics identified, and an analysis of heuristic sub-component performance. Section 4.6 provides conclusions and possible directions for future work.

4.2 Robust optimisation

4.2.1 Problem description

A general optimisation problem without consideration of uncertainty takes the form:

$$\begin{aligned} &\min f(\boldsymbol{x}) \\ &\text{s.t. } \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

This is the nominal problem. The objective $f : \mathbb{R}^n \to \mathbb{R}$ operates on the n-dimensional vector of decision variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ in the feasible region $\mathcal{X} \subseteq \mathbb{R}^n$. Here we assume box constraints $\mathcal{X} = \prod_{i \in [n]} [l_i, u_i]$. Any other feasibility constraint is assumed to be dealt with by a penalty in the objective. The notation $[n] := \{1, \ldots, n\}$ is used. Consider the problem due to [Kru12] in Figure 4.1, where $\mathcal{X} \subseteq \mathbb{R}^1$, $l_1 = 0$ and $u_1 = 10$. The nominal problem is the black curve.

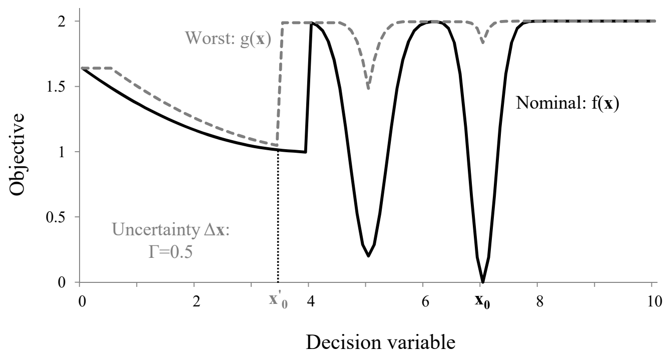


Figure 4.1: The worst case cost curve (dashed grey) is generated by determining the maximum objective value in the uncertainty neighbourhood around all points $\boldsymbol{x}$ on the nominal (solid black) curve. Due to the uncertainty the global optimum shifts to $\boldsymbol{x}_0'$.

Introducing uncertainty $\Delta\boldsymbol{x}$ around the intended solution $\boldsymbol{x}$, makes only a solution $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \Delta\boldsymbol{x}$ achievable. If it is assumed that the uncertainty neighbourhood around a candidate is completely defined by a radius $\Gamma > 0$, the uncertainty set is [BNT10b]:

$$\mathcal{U} := \{\Delta\boldsymbol{x} \in \mathbb{R}^n \mid \|\Delta\boldsymbol{x}\| \leq \Gamma\}$$

where $\|\cdot\|$ represents the Euclidean norm. Using a local maximisation to find a robust solution $\boldsymbol{x}$, the worst case value $g(\boldsymbol{x})$ is optimised for any $\tilde{\boldsymbol{x}}$ in the uncertainty neighbourhood of $\boldsymbol{x}$:

$$g(\boldsymbol{x}) := \max_{\Delta\boldsymbol{x}\in\mathcal{U}} f(\boldsymbol{x} + \Delta\boldsymbol{x})$$

In Figure 4.1, $\Gamma = 0.5$ and each point on the worst case cost (dashed grey) curve $g(\boldsymbol{x})$ is generated by finding the maximum value on the nominal curve within a range of -0.5 to $+0.5$ of the desired solution $\boldsymbol{x}$.

The complete min max robust problem then involves finding the outer minimum objective in $\mathcal{X}$, where that objective is itself an inner maximisation in the uncertainty neighbourhood around each solution $\boldsymbol{x} \in \mathcal{X}$ for the nominal objective function:

$$\min_{\boldsymbol{x}\in\mathcal{X}} g(\boldsymbol{x}) = \min_{\boldsymbol{x}\in\mathcal{X}} \max_{\Delta\boldsymbol{x}\in\mathcal{U}} f(\boldsymbol{x} + \Delta\boldsymbol{x}) \tag{MinMax}$$

In the example this moves the global minimisation search from the black curve to the grey curve, where the global optimum shifts from $\boldsymbol{x}_0$ to $\boldsymbol{x}_0'$.

As $\boldsymbol{x} + \Delta\boldsymbol{x}$ may be outside of $\mathcal{X}$, here it is not assumed that f is restricted to $\mathcal{X}$. If it is required that $\boldsymbol{x} + \Delta\boldsymbol{x} \in \mathcal{X}$ for all $\Delta\boldsymbol{x} \in \mathcal{U}$, this could be achieved for example through the reduction of the original $\mathcal{X}$ by Γ.

4.2.2 State of the art

Since its initial formalisation [KY97, BTN98] robust optimisation has been heavily aligned with mathematical programming, see [BBC11, GMT14, GS16]. Where mathematical programming techniques cannot be applied, metaheuristics may be considered.

However only limited consideration has been given to robust black-box optimisation under implementation uncertainty, see [MWPL13, GS16, MWPL16].

Whilst standard metaheuristics can be extended to the robust worst case through the brute force addition of an inner maximisation routine into an outer minimisation setting e.g. [HGW19, HGD20b], more refined robust-specific methods may be desirable. Such techniques include co-evolutionary approaches [Her99, SK02, Jen04, CSZ09, MKA11], robust evolutionary approaches [TG97, BS07], and the use of emulation by surrogates or meta-models [ONL06, BS07, ZZ10, KVDHL16] including Kriging [MWPL13, uRLvK14, MWPL16, uRL17] and Bayesian techniques [CLSS17, SEFR19]. However specific assumptions or simplifications are typically required for such methods to be effective, or there are limitations on the problems that can be addressed e.g. due to dimensionality.

Two existing general robust approaches requiring no further assumptions or simplifications are given specific attention here. They form the basis for some of the algorithmic building blocks which constitute the grammar in our GP analysis. These are the local descent directions (d.d.) approach [BNT07, BNT10a, BNT10b] and the global largest empty hypersphere (LEH) method [HGW19]. Both are single-solution techniques, although elements of these approaches have been incorporated into robust population based approaches [HGD20b]. First, however, we consider the non-robust particle swarm optimisation (PSO) metaheuristic [KE95, KES01, Tal09], which is the basis for all heuristics in our GP search. Constituent elements of a typical PSO algorithm are included as building blocks in our GP grammar.

4.2.2.1 Particle swarm optimisation

PSO is a population based approach which moves a 'swarm' of particles through points in the decision variable space, performing function evaluations and iterating particle positions through particle level 'velocity' vectors. Velocities are based on particle histories, shared information from the swarm, scaling, and randomisation. The intention is for the behaviour of this complex systems of particles to approximate a global optimisation search. There are very many PSO formulations building on this general concept, see for example [Kam09, NMES11, ZWJ15, Kir17, SBP18].

In a robust setting two-swarm co-evolutionary PSO techniques have been considered [SK02, MKA11], whilst [HGW19] employ a simple robust PSO (rPSO) as a comparator heuristic, with inner maximisation by random sampling. The rPSO from [HGW19] is extended in [HGD20b] through the addition of d.d. and LEH elements. Here the framework for each heuristic in the GP population is a basic PSO formulation [KE95, KES01, Tal09], built upon through the availability of more complex algorithmic building blocks in our grammar.

In a basic non-robust PSO formulation, the swarm (population) of N particles start at iteration $t = 0$ randomly located at points $\boldsymbol{x}^j(0)$ in $\mathcal{X}$, where the function is evaluated;

here $j = 1, \ldots, N$. Each particle stores information on the best position it has visited in its history, $\boldsymbol{x}_*^j$. Best refers to the lowest objective function value.

Information sharing is a key element of PSO, with each particle associated with a neighbourhood of other particles. Within a neighbourhood information is shared on the best point visited by any neighbourhood particle in their entire histories, $\hat{\boldsymbol{x}}_*$. A number of neighbourhood topologies are included as components in the grammar here, see Section 4.4.2.4.

A particle is moved to a location $\boldsymbol{x}^j(t)$ at iteration t, through the addition of that particle's current velocity vector $\boldsymbol{v}^j$ to its previous position:

$$\boldsymbol{x}^j(t) = \boldsymbol{x}^j(t-1) \; + \; \boldsymbol{v}^j(t) \tag{PSOmove}$$

There are a number of alternative velocity formulations. In the grammar in Section 4.4.2 two of the most basic formulations are considered, including so-called inertia [SE98, KES01] and constriction [CK02, KES01] coefficients:

$$\boldsymbol{v}^j(t) = \omega \cdot \boldsymbol{v}^j(t-1) \; + \; C_1 \cdot \boldsymbol{r}_1 \cdot (\boldsymbol{x}_*^j - \boldsymbol{x}^j(t-1)) \; + \; C_2 \cdot \boldsymbol{r}_2 \cdot (\hat{\boldsymbol{x}}_* - \boldsymbol{x}^j(t-1)) \tag{Inertia}$$

$$\boldsymbol{v}^j(t) = \chi \cdot \left(\boldsymbol{v}^j(t-1) + C_1 \cdot \boldsymbol{r}_1 \cdot (\boldsymbol{x}_*^j - \boldsymbol{x}^j(t-1)) + C_2 \cdot \boldsymbol{r}_2 \cdot (\hat{\boldsymbol{x}}_* - \boldsymbol{x}^j(t-1)) \right) \tag{Constriction}$$

where

$$\chi = \frac{2}{\left| 2 - \phi - \sqrt{\phi^2 - 4\phi} \right|}$$

with

$$\phi = C_1 + C_2$$

Here particle velocities $\boldsymbol{v}^j(0)$ are initialised by uniform random sampling $\sim U(0\,,\,0.1)^n$ [Eng12]. Each component of the random vectors $\boldsymbol{r}$ is typically randomly sampled individually, $\boldsymbol{r}_1\,,\,\boldsymbol{r}_2 \sim U(0\,,\,1)^n$. Vector multiplication is component-wise. The scalar terms C_1, C_2 represent weightings that a particle puts on its $\boldsymbol{x}_*^j$ (C_1) versus $\hat{\boldsymbol{x}}_*$ (C_2) location data. The inertia scalar ω moderates the significance of the preceding velocity, whilst the constriction scalar χ is used to avoid particles 'exploding' – disappearing out of the feasible region. Here an invisible boundary condition is adopted [RR04], with particles allowed to leave the feasible region but no function evaluations undertaken for particle locations outside of $\mathcal{X}$.

A non-robust PSO can be extended to a robust approach through the addition of an inner maximisation search component. This is the approach adopted here. The inner maximisation techniques available as grammar components are discussed in Section 4.4.2.6.

The (Inertia) and (Constriction) formulations represent an rPSO baseline movement capability. For any given heuristic in our GGGP the movement calculation can be extended through the addition of components, described in Section 4.4.2. This includes building blocks based on a series of metaheuristics for robust problems developed using rPSO as a framework in [HGD20b], and novel features here. These developments are largely based around two robust search techniques, d.d. and LEH.

4.2.2.2 Descent direction

Descent directions is an exploitation-focussed, individual-based robust local search technique [BNT07, BNT10b, BNT10a] for solving (MinMax), although it can easily be extended to approximate a global search through random re-starts each time a local search completes [HGW19]. We briefly summarise the method outlined in [BNT07, BNT10b, BNT10a].

In d.d. at each candidate point $\boldsymbol{x}$ in the decision variable space that the search moves to, an inner maximisation search is performed to assess that point's uncertainty neighbourhood $N(\boldsymbol{x}) = \{\boldsymbol{x} + \Delta\boldsymbol{x} \mid \Delta\boldsymbol{x} \in \mathcal{U}\}$ and approximate the worst case cost $\tilde{g}(\boldsymbol{x}) \approx g(\boldsymbol{x})$. Function evaluations are stored in a history set H, and at each candidate the local information is further exploited through the identification of poor 'high cost' points (hcps), those with the greatest objective function value, in H and within the Γ-radius uncertainty region. At a candidate point $\boldsymbol{x}$ the high cost set $H_\sigma(\boldsymbol{x})$ is defined as:

$$H_\sigma(\boldsymbol{x}) := \{\boldsymbol{x}' \in H \cap N(\boldsymbol{x}) \mid f(\boldsymbol{x}') \geq \tilde{g}(\boldsymbol{x}) - \sigma\}$$

σ is the threshold value for determining what constitutes an hcp.

The optimum (descent) direction originating at the current candidate $\boldsymbol{x}(t)$ at step t, and pointing away from the hcps, is then calculated using mathematical programming. The angle θ between the vectors connecting the points in $H_\sigma(\boldsymbol{x}(t))$ to $\boldsymbol{x}(t)$, and $\boldsymbol{d}$, is maximised:

$$\min_{\boldsymbol{d},\beta} \ \beta \tag{Soc}$$

$$\text{s.t.} \ \|\boldsymbol{d}\| \leq 1 \tag{Con1}$$

$$\boldsymbol{d}^T \left(\frac{\boldsymbol{h} - \boldsymbol{x}(t)}{\|\boldsymbol{h} - \boldsymbol{x}(t)\|} \right) \leq \beta \qquad \forall \boldsymbol{h} \in H_\sigma(\boldsymbol{x}(t)) \tag{Con2}$$

$$\beta \leq -\varepsilon \tag{Con3}$$

Here ε is a small positive scalar, so from (Con3) β is negative. The left hand side of constraint (Con2) is the multiplication of $\cos\theta$ and $\|\boldsymbol{d}\|$, for all hcps in $H_\sigma(\boldsymbol{x}(t))$ and a feasible direction $\boldsymbol{d}$. (Con2) therefore relates β to the maximum value for $\cos\theta$ across all hcps. As the objective (Soc) is to minimise β, and β is negative, the angle θ will be greater than 90^o and maximised. Also minimising β in combination with (Con1) normalises $\boldsymbol{d}$. A standard solver such as CPLEX can be used to solve (Soc). When a feasible direction cannot be found, that is (Soc) cannot be solved, the algorithm stops: a robust local minimum has been reached.

The local search repeats at step t by moving away from the current candidate $\boldsymbol{x}(t)$, in this optimum direction $\boldsymbol{d}$ with a step size $\rho(t)$ large enough that the points in $H_\sigma(\boldsymbol{x}(t))$ are at a minimum on the boundary of the uncertainty neighbourhood of the next candidate at step $t+1$. Then $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \rho(t) \cdot \boldsymbol{d}$, where:

$$\rho(t) = \min \left\{ \boldsymbol{d}^T(\boldsymbol{h} - \boldsymbol{x}(t)) + \sqrt{(\boldsymbol{d}^T(\boldsymbol{h} - \boldsymbol{x}(t)))^2 - \|\boldsymbol{h} - \boldsymbol{x}(t)\|^2 + \Gamma^2} \mid \boldsymbol{h} \in H_\sigma(\boldsymbol{x}(t)) \right\} \tag{Rho}$$

Steps are repeated until a local minimum is reached.

In one of the rPSO variants described in [HGD20b], given neighbourhood uncertainty information for each particle at each step, a descent direction vector is calculated. This vector $\boldsymbol{d}^j(t-1)$ for particle j at step t is used for the calculation of an additional velocity component:

$$C_3 \cdot \boldsymbol{r}_3 \cdot \boldsymbol{d}^j(t-1) \tag{ddVel}$$

In [HGD20b] each component of $\boldsymbol{r}_3$ is randomly sampled individually, $\boldsymbol{r}_3 \sim U(0\ ,\ 1)^n$, vector multiplication is component wise, and the scalar term C_3 represents a weighting that a particle puts on its local descent direction vector. From step $t = 1$ onwards a variant on the baseline PSO (Inertia) velocity formulation is then used in [HGD20b]:

$$\boldsymbol{v}^j(t) = \omega \cdot \boldsymbol{v}^j(t-1) + C_1 \cdot \boldsymbol{r}_1 \cdot (\boldsymbol{x}^j_* - \boldsymbol{x}^j(t-1)) + C_2 \cdot \boldsymbol{r}_2 \cdot (\hat{\boldsymbol{x}}_* - \boldsymbol{x}^j(t-1)) + C_3 \cdot \boldsymbol{r}_3 \cdot \boldsymbol{d}^j(t-1) \tag{InertiaV2}$$

Building blocks components based on this d.d. approach, and associated parameters, are considered in the grammar here. Details are given in Section 4.4.2.

4.2.2.3 Largest empty hypersphere

Largest empty hypersphere is an exploration-focussed individual-based robust global search technique [HGW19] for solving (MinMax). LEH takes the d.d. concept of hcps to a global setting, identifying a high cost set H_τ of poor points from within the global history set H, and moving to the centre of the region completely devoid of all such points. H_τ contains those points in H with nominal objective function value $f(\boldsymbol{x})$ greater than a threshold τ. In LEH τ equals the current estimated robust global minimum value.

The centre of the LEH, $\boldsymbol{x}(t) \in \mathcal{X}$ at iteration t, is the estimated point furthest from all hcps in H_τ, and is approximated using a genetic algorithm (GA). Movement from centre of LEH to centre of LEH repeats until no point $\boldsymbol{x}(t) \in \mathcal{X}$ which is at least Γ away from all hcps can be identified, or a defined budget of available objective function evaluations (model runs) is exhausted. The final estimate for the global robust minimum is accepted.

A key feature of LEH is the early stopping of neighbourhood searches at any candidate where an improved estimated robust global optimum cannot be achieved. In theory an inner maximisation is performed at each candidate $\boldsymbol{x}(t)$, however in LEH each objective function evaluation in a neighbourhood analysis $f(\boldsymbol{x}(t) + \Delta\boldsymbol{x}(t))$ is compared to τ, with the inner search terminating if that value exceeds τ. This recognises that the current point $\boldsymbol{x}(t)$ won't improve on the estimated robust global optimum, and has the potential to afford considerable savings in local function evaluations and so enable a more efficient exploration of $\mathcal{X}$.

One of the rPSO variants in [HGD20b] is based around core elements of the LEH approach. Firstly the stopping condition is employed at a particle level for each particle in each iteration. For any particle j an inner maximisation search may begin but is terminated early if an uncertainty neighbourhood point exceeds the particle best information $\boldsymbol{x}_*^j$. In fact by first assessing the complete history set H of all previous function evaluations, no inner maximisation may be necessary if it is determined that some historical value in the particle's uncertainty neighbourhood already exceeds the best information $\boldsymbol{x}_*^j$.

Using a second novel LEH-based feature, particles are assessed for 'dormancy', defined as a state where no function evaluations have been required by a particle for a specified number of iterations. This may be due to the repeated identification of existing neighbourhood points which exceed the particle's best information $\boldsymbol{x}_*^j$, prior to undertaking an inner maximisation. Or it may be due to the particle repeatedly moving outside the feasible region, linked to the use of an invisible boundary condition [RR04]. In either case dormancy suggests that a particle has become 'stuck'. In [HGD20b] dormant particles are relocated to the centre of the largest empty hypersphere devoid of all hcps, using the current robust global minimum as the high cost threshold. More details of the LEH-based components and associated parameters available in the grammar here are given in Section 4.4.2.

4.3 The automatic generation of heuristics

In seeking to develop improved metaheuristics for robust problems an obvious question is what features should be included in the search methodology. This is a step beyond the issues of what existing search technique a decision maker might employ, or what parameter settings might be used for any given problem. These issues impact the effectiveness of any optimisation search.

Given a problem for which an optimisation search is to be undertaken, the field of hyper-heuristics encompasses techniques which employ a search methodology to automatically identify or generate heuristics for application to that problem. The hyper-heuristic itself does not search the problem solution space, but rather seeks a heuristic for application to the problem. A high level classification of hyper-heuristic approaches distinguishes between methods for selecting a heuristic, from a space of heuristics, and methods for generating a heuristic [BHK$^+$09, BGH$^+$13]. Our interest is in the latter.

The automatic generation of a search heuristic is a specific application of the broader theme of the automatic generation of algorithms, or the automatic generation of computer programs. One technique which can be applied in the general case and to the specific issue of automatically generating a search heuristic is genetic programming (GP) [Koz92, BHK$^+$09]. This employs the well known high level concepts of selection, combination and mutation to evolve a population of computer programs, or in our case search heuristics for robust problems.

When considering the applications of genetic programming to automatically generate optimisation search approaches [BGH$^+$13], the use of tree-based context-free grammar-guided GP [MHW$^+$10] to the generation of PSO heuristics described in [MP16, MP17] is of particular interest here. We adopt that approach and apply it to PSO based heuristics for robust problems.

Relatively little work has been undertaken on the application of GP to optimisation search techniques for uncertain problems. One example from the field of stochastic optimisation is [MZ18], where GP is applied to a vehicle routing problem including uncertainty. In terms of optimisation for robust problems, [GH19] use a simple GP-based approach to evolve techniques for robust combinatorial optimisation problems. However, to the author's knowledge [GH19] is the only explicit use of a GP-based approach applied to a robust problem, and there is no application of GP to metaheuristics for black-box robust optimisation problems prior to the work outlined here.

4.4 The genetic programming of metaheuristics for robust problems

4.4.1 Genetic programming

Our aim is to develop improved metaheuristics for robust problems and remove the manual determination of feature-technique-parameter choices, through the automatic generation of algorithms by genetic programming [Koz92, MHW$^+$10, Noh11]. GP is an evolutionary process, and here each individual in the GP population is a heuristic. For an initial population of heuristics, a measure of fitness is calculated for each individual. A new generation of heuristics is then determined through typical evolutionary algorithm fitness-based selection, combination and mutation processes. This repeats over multiple generations, at the end of which the fittest heuristic is chosen.

Each heuristic in the GP process is made up of multiple algorithmic sub-components and their parameter settings, which when combined appropriately form executable search heuristics. So the GP solution space consists of sub-components and their parameters.

We define algorithmic sub-components along with the production rules which determine how they combine to form complete heuristics. Sub-components are designed to integrate effectively under those construction rules. This is our grammar [Koz92], which is employed within an evolutionary framework. That framework must be capable of performing combination and mutation operations on heuristics constructed from building blocks. Here a tree-based GP evolutionary process is used to facilitate these processes, as described in Section 4.4.3. Details of the individual sub-components are given now in Section 4.4.2.

4.4.2 Grammar

4.4.2.1 Structure

The heuristics considered here consist of outer minimisation and inner maximisation searches, wrapped around a black-box model. Each model run generates a single objective function output corresponding to a point in the model decision variable space $\mathcal{X}$. We use a PSO frame for all heuristics, comprising a swarm of particles moving over a series of iterations. This constitutes the outer minimisation, with inner maximisations undertaken at the particle level to determine the robust objective function value at a point in $\mathcal{X}$.

Every member of the population in the GP analysis has the same basic algorithmic structure, described by Algorithm 7. We assume a limit on the number of function evaluations (model runs) available. The swarm is initialised randomly, and the defined form of inner maximisation undertaken to determine robust objective values for each particle. Particle movement is then controlled by the velocity equation formulation and the forms of topology and movement, i.e. how particle velocities are calculated, how particles share information, and how these elements are used. The swarm moves and particle level inner maximisations are undertaken again. This repeats until the budget is exhausted. On completion the current best estimate for robust global minimum is accepted.

Generating metaheuristics for robust problems in a GP process requires the definition of a grammar: algorithmic sub-components and the rules for combining them. Here the high level outline of each heuristic, Algorithm 7, also forms the high level design criteria in the grammar: the outer minimisation layer as a swarm of particles, some movement formulation, a topology dictating particle information-sharing, and an inner maximisation layer.

Our sub-components and the production rules for generating complete heuristics are defined in the grammar in Figure 4.2. The specific approach adopted here is known as context-free grammar genetic programming (CFG-GP). This uses a tree-based representation of algorithms and standard tree-based operators in the evolutionary process, see [MHW$^+$10, MP16].

The grammar includes non-terminal nodes, indicated by `< >`, terminal nodes, and the production rules (`::=`). The generation of a heuristic begins at the `<Start>` node, resulting in the generation of more nodes by following the rules in Figure 4.2. Each non-terminal node leads to the generation of further nodes according to the production rules, with each non-terminal node expanded upon until a terminal node is reached. The result is the generation of a series of nodes corresponding to elements of the heuristic. Non-terminal nodes do not result in the generation of further nodes, but instead in the determination of parameter settings: parameter values or choices of individual sub-components. On reaching a non-terminal node that portion of the heuristic is complete. The final heuristic is achieved when there are no more non-terminal nodes to expand upon in the sequence.

Algorithm 7 Overview of a robust particle swarm optimisation algorithm

```
Inputs: Swarm $size$, extent of inner search ($inExt$), $budget$ of function evaluations
Parameters: Form of $inner$, form of $topology$, form of $velocity$, form of $movement$
Parameters: $innerParams$, $topolParams$, $velParams$, $moveParams$

1: $t \leftarrow 0$
2: while ($budget > 0$) do
3:     for all ($j$ in $1, \ldots, size$) do
4:         if ($t = 0$) then
5:             Randomly initialise particle $\boldsymbol{x}^j(0) \in \mathcal{X}$
6:         else
7:             Update particle velocity according to ($velocity, velParams$)
8:             Update particle position according to
9:                 ($movement, moveParams, topology, topolParams$)
10:        end if
11:        if ($\boldsymbol{x}^j(t) \in \mathcal{X}$) then
12:            Perform inner maximisation:
13:            for all ($k$ in $1, \ldots, inExt$) do
14:                Select uncertainty neighbourhood point: ($inner, innerParams$)
15:                Evaluate function (run model, generate objective)
16:                $budget \leftarrow budget - 1$
17:                if ($budget = 0$) then break: goto end end if
18:            end for
19:        end if
20:    end for
21:    $t \leftarrow t + 1$
22: end while
23: return Current estimate of robust global best
```

Sub-component details are given in Sections 4.4.2.2 to 4.4.2.9. The high level `<Outer>` and `<Inner>` elements are identifiable in Figure 4.2. `<Outer>` consists of `<Group>` (swarm size), `<Mutation>`, `<Network>` and `<Capability>` elements. `<Mutation>` refers to random variations applied to a particle's next location, `<Network>` specifies the rules for

```
< Start >::= < Outer > < Inner >
< Outer >::= < Group > < Mutation > < Network > < Capability >
< Group >::= Uniform [2, 50]
< Mutation >::= < Mutate > < Prob Mutate >
< Mutate >::= None | Uniform | Gaussian
< Prob Mutate >::= Uniform [0, 0.5]
< Network >::= Global | Focal | Ring (n=2) | 2D von Neumann | Clan | Cluster | Hierarchical
< Capability >::= < Baseline > < Movement >
< Baseline >::= Inertia | Constriction
< Inertia >::= < C1 > < C2 > < ω >
< Constriction >::= < C1 > < C2 >
< C1 >::= Uniform [0, 2.4]
< C2 >::= Uniform [0, 2.4]
< ω >::= Uniform [0.1, 0.9]
< Movement >::= { } | < DD > | < LEH > | < DD > < LEH >
< DD >::= < C3 > < σ > < σ limit > < Min step > < r3 >
< C3 >::= Uniform [0, 10]
< σ >::= Uniform [0.1,0.4]
< σ limit >::= Uniform [0.001, 0.01]
< Min step >::= Uniform [0.001,0.1]
< r3 >::= < Random r3 > | 1
< Random r3 >::= Uniform [0, 1]
< rndLEH >::= < LEH > | Random relocation
< LEH >::= < lpop > < lmutP > < lmutA > < lelites > < ltour > < Dorm >
< lpop >::= Uniform (4, 5, 10, 20, 25)
< lmutP >::= Uniform [0, 1]
< lmutA >::= Uniform [0, 0.5]
< lelites >::= Uniform [1, 3]
< ltour >::= Uniform [0, 0.5]
< Dorm >::= Uniform [1, 5]
< Inner >::= < In Ext > < Form Inner > < nDorm > < nPBest > < Stopping >
< In Ext >::= Uniform [0, 1]
< Form Inner >::= { } | < In PSO > | < In GA >
< In PSO >::= < In Swarm > < In C1 > < In C2 > < In ω >
< In Swarm >::= Uniform [0, 1]
< In C1 >::= Uniform [0, 2.4]
< In C2 >::= Uniform [0, 2.4]
< In ω >::= Uniform [0.1, 0.9]
< In GA >::= < In pop > < In mutP > < In mutA > < In elites > < In tour >
< In pop >::= Uniform [0, 1]
< In mutP >::= Uniform [0.01, 0.5]
< In mutA >::= Uniform [0.01, 0.5]
< In elites >::= Uniform [0, 1]
< In tour >::= Uniform [0, 1]
< nDorm >::= No | Yes
< nPBest >::= No | Yes
< Stopping >::= No | Yes
```

Figure 4.2: Context-free grammar employed here for the construction of metaheuristics for robust problems. The symbol | designates a choice of one of the alternatives.

particle information sharing, and `<Capability>` covers a number of sub-components which combine to form the rules for particle movement. `<Capability>` breaks down into `<Baseline>` and `<Movement>`, where `<Baseline>` refers to core PSO velocity equations, and `<Movement>` refers to extended capabilities built around d.d. [BNT10b] and LEH [HGW19] techniques and their variants [HGD20b]. `<Inner>` is by random sampling, or a PSO or GA search, along with additional `<nDorm>`, `<nPBest>` and `<Stopping>` capabilities, based on features in [HGD20b] and explained here in Sections 4.4.2.7 to 4.4.2.9. There are also a number of sub-components and parameters associated with many of these elements, which in total constitutes our grammar.

One way to visualise this process is in the form of a tree [Koz92, Whi95], Figure 4.3, showing the high level structure of a heuristic generated by the production rules in Figure 4.2. `<Start>` produces the non-terminal nodes `<Outer>` and `<Inner>`. `<Outer>` is expanded upon, and when it is complete the `<Inner>` node is returned to. From the `<Outer>` node `<Group>`, `<Mutation>`, `<Network>` and `<Capability>` are generated one at a time, fully expanding on `<Group>` before moving to `<Mutation>` and so on. When `<Capability>` is complete `<Outer>` is complete.

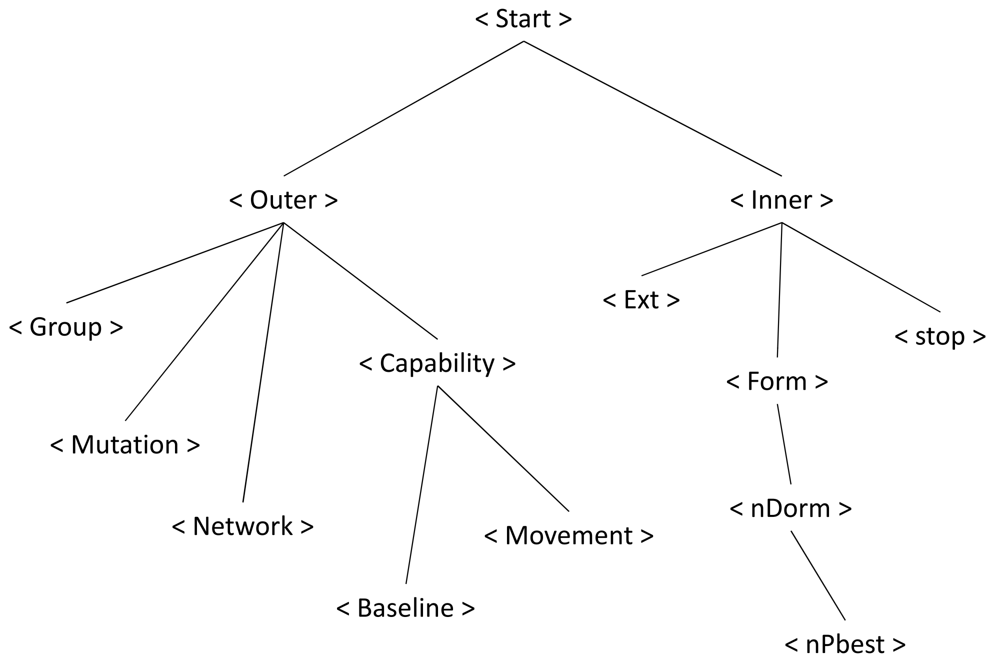


Figure 4.3: Solution representation: high level tree-based representation of the heuristic generated by following the CFG-GP grammar production rules in Figure 4.2.

`<Group>` generates a terminal node, a randomly sampled value between 2 and 50 for the number of particles (swarm size) in the heuristic. Having reached a terminal node, the next non-terminal node in the chain generated so far but not yet expanded upon, is `<Mutation>`. This refers to the mutating of individual particle positions. `<Mutation>` leads to the non-terminal combination of `<Mutate>` and `<Prob Mutate>`, both leading to terminal choices, respectively either `None` (no mutation), or `Uniform` or `Gaussian` mutation, and if mutation the probability of mutation from the range 0 to 0.5. The symbol | in the production rules designates a choice of one of the alternatives. On randomly choosing a mutation alternative and probability, if required, the next non-terminal node in the chain, `<Network>`, is returned to. And so on.

4.4.2.2 Building blocks: Particle swarm framework

A basic PSO extended by an inner maximisation forms the basis for all heuristics here. Core PSO elements are a `<Group>` (swarm) of particles, a `<Baseline>` velocity equation of either `Inertia` or `Constriction` forms described in Section 4.2.2.1, and some system of particle information sharing. The latter, `<Network>`, is discussed in Section 4.4.2.4. In all heuristics the `<Group>` size, `<C1>` and `<C2>` parameter values are sampled from the ranges defined in the grammar in Figure 4.2. The need for the <ω> term depends on the choice of `<Baseline>`.

4.4.2.3 Building blocks: Mutation

The use of the non-terminal node `<Mutation>` is considered at a particle level, after the candidate position in the next iteration has been determined, see [MP17]. If used, mutation is considered separately for each particle and at the dimensional level, as a final stage in the movement calculation. For each particle it is determined whether or not to mutate by sampling against the probability `<Prob Mutate>`. If mutation is confirmed, any given dimension is mutated with probability randomly sampled from between 0 and $1/n$, so on average only one dimension is changed. The magnitude of change is sampled from either the Uniform or Gaussian distributions as appropriate, and related to the dimensional bounds.

4.4.2.4 Building blocks: Networks

The sharing of information throughout the swarm to inform movement at the individual particle level, is a core PSO element. Here that form of sharing between particles is determined by the `<Network>` component. Of the large number of networks available [KE95, KM02, MKN03, JM05, dCBF09, WYO16, MP17], we consider seven alternatives. Each particle is assigned to a network neighbourhood. At each iteration information on the best neighbourhood point visited by any particle in the network across all iterations, $\hat{\boldsymbol{x}}_*$, is shared in a manner defined by the baseline velocity equations (Inertia) or (Constriction).

- *Global*: This is the most basic formulation, with all particles accessing the same neighbourhood information – the current robust global minimum location [KE95, KES01].
- *Focal*: A singe particle is randomly selected as the focal. All particles access the same neighbourhood information, the focal particle's best information, [KES01, KM02].

- *Ring (size=2)*: In a network sense all particles may be randomly arranged into a ring formation. With this topology a particle has access to the best information from the adjoining particles in the ring. Here we set the size equal to two, so a particle has access to its two neighbour's (one either side in the ring formation) best information [KES01, KM02].

- *2D von Neumann*: In a network sense particles may be randomly arranged into a 2D grid, or more correctly the surface of a torus where the grid wraps around so that the top and bottom join, as do the left and right hand sides. Each particle has four neighbours, the nearest north, south, east and west particles in this grid formation, accessing the best information in this neighbourhood [KM02, MP17].

- *Clan*: Each particle is randomly placed in a network sub-group, or clan. Each clan is linked to each other clan via a clan leader. The leader in each clan is the particle with the best performance, so leaders may change over iterations of the swarm. Each leader shares their information with all other clans [dCBF09, MP17].

- *Cluster*: Each particle is randomly placed in a network sub-group, or cluster. Within each cluster a number of 'informant' particles are randomly assigned. The number of informants is one less than the number of clusters, and within each cluster one informant is linked to one other cluster. Informants remain fixed. Within a cluster the best information is shared between all particles. Informant particles share their information with the single cluster they link to [MKN03, MP17].

- *Hierarchical*: All particles are randomly arranged in a tree formation, in a network sense. The depth and width of the tree is dependent on the number of particles (swarm size). Each particle communicates with the particle above it in the tree. At each iteration of the swarm the positions in the tree can shift: if a particle below another one in the tree performs better, the two swap positions. This applies to all particles in each iteration [JM05, MP17].

4.4.2.5 Building blocks: Additional movement capability

The baseline PSO capability can be augmented by additional movement components based on the descent directions [BNT10b] and largest empty hypersphere [HGW19] approaches, as proposed in [HGD20b]. The additional formulations available in our grammar are none { }, a d.d. based approach `<DD>`, an LEH based approach `<LEH>`, or a combined d.d. and LEH based approach `<DD> <LEH>`. In the case of { } just the rPSO `<Baseline>` formulation is used. Otherwise the rPSO d.d. or LEH approaches, or both, augment the baseline rPSO formulation at the particle level as described in Sections 4.2.2.2 and 4.2.2.3, and [HGD20b].

Both `<DD>` and `<LEH>` require the determination of further non-terminal nodes. `<DD>` employs the nodes: `<C3>`, <σ>, <σ `limit`>, `<Min step>` and `<r3>`. These are d.d.

parameters whose descriptions can be found in [BNT10b, HGW19, HGD20b]. They all terminate once parameter values have been determined, with the exception of `<r3>` which relates to the additional C_3 component in the d.d. equations (ddVel) and (InertiaV2) in Section 4.2.2.2. In the original formulation each element of $\boldsymbol{r}_3$ is randomly sampled individually, $\boldsymbol{r}_3 \sim U(0\,,1)^n$. This alternative is available in the component `<Random r3>` in the grammar, along with another where each element of $\boldsymbol{r}_3$ is set to unity. The latter is a recognition that a locally calculated d.d. vector might be more effective without added random variation.

`<LEH>` relates to the relocation of a particle deemed 'dormant', and requires the determination of either the non-terminal node `<LEH relocation>` or the terminal selection of `Random relocation`. In the original formulation a particle is moved to the centre of the LEH devoid of all identified high cost points [HGD20b], as described in Section 4.2.2.3. Here this is designated by `<LEH relocation>`, and if selected the parameters `<lpop>`, `<lmutP>`, `<lmutA>`, `<lelites>`, `<ltour>` and `<Dorm>` must be determined. In the grammar all of these parameters terminate once their values have been generated; their descriptions can be found in [HGW19, HGD20b]. However an alternative is available here, `Random relocation`, which as the name suggests simply relocates a particle randomly in $\mathcal{X}$. This does not use any additional parameters.

4.4.2.6 Building blocks: Inner maximisation

In theory an inner maximisation search is required to accurately estimate the worst objective function value in a candidate point's uncertainty neighbourhood. In practice, issues such as the run time for each function evaluation (model run) may be prohibitive. Here we assume a limit on the number of function evaluations that are possible. This will likely restrict the accuracy of any search, as it will cause some trade-off between the extent of an inner search (robustness) and the level of global exploration. Such a trade-off is not simple [MLM15, EDHX17]. In this context the choice of approach for, and the extent of, the inner maximisation is not obvious. Here the non-terminal `<Inner>` node generates several further non-terminal nodes: `<In Ext>`, `<Form Inner>`, `<nDorm>`, `<nPBest>` and `<Stopping>`.

`<nDorm>`, `<nPBest>` and `<Stopping>` are discussed in Sections 4.4.2.7 to 4.4.2.9. `<In Ext>` and `<Form Inner>` relate to the extent and form of inner search. `<In Ext>` is based on a randomly sampled value in the range 0 to 1. This value is related to the outer particle group size and budget of function evaluations, to determine a corresponding integer size of inner search. There are three alternatives for `<Form Inner>`: random sampling { }, or inner PSO `<In PSO>` or genetic algorithm `<In GA>` searches. All apply to a candidate point's Γ-radius uncertainty neighbourhood, Section 4.2.1. If random sampling is used no additional parameters are required. Multiple parameter nodes are required for either `<In PSO>` or `<In GA>`.

`<In PSO>` requires the determination of `<In Swarm>`, `<In C1>`, `<In C2>` and `<In` ω`>`, the parameters for an (Inertia) form of PSO: an inner swarm size and settings for C_1, C_2 and ω. For an inner PSO the (Inertia) formulation is fixed. `<In GA>` requires the determination of `<In pop>`, `<In mutP>`, `<In mutA>`, `<In elites>` and `<In tour>`, parameters for a standard form of GA [Tal09]: an inner population size, and settings for the probability of and amount of mutation, the number of elites, and a tournament size. If employed, `<In Swarm>` or `<In pop>` are initially determined in the range 0 to 1, and then related to `<In Ext>` to give a corresponding integer value. For an inner GA, `<In elites>` and `<In tour>` are initially determined in the range 0 to 1, then related to `<In pop>` to give integer values.

4.4.2.7 Building blocks: Dormancy – use of neighbourhood information

The consideration of particle dormancy leading to its relocation [HGD20b], is described in Section 4.2.2.3. Dormancy refers to a particle becoming 'stuck'. Of concern here is when this might be due to the particle being in an already identified poor region of the solution space, and thereby repeatedly not requiring any function evaluations. In our grammar the determination of dormancy for each particle in each generation may (Yes) or may not (No) make use of the history set H of all points evaluated, and specifically those points within a particle's uncertainty neighbourhood. The choice is represented in node `<nDorm>`.

4.4.2.8 Building blocks: Supplement $\boldsymbol{x}_*^j$ – neighbourhood information

A particle's robust value is based on an inner search, and leads to the determination of the particle's personal best location $\boldsymbol{x}_*^j$ as employed in the (Inertia) or (Constriction) velocity formulation. Given a completed inner search, if relevant the identified robust value can be updated by the worst point already identified in the history set H within the particle's current uncertainty neighbourhood. The choice of whether (Yes) or not (No) historic information is used in this way is included as a component here, represented by node `<nPBest>`.

4.4.2.9 Building blocks: Stopping condition

The use of a stopping condition in an inner search for a given particle, if a point is identified with objective function value exceeding that particle's personal best information, has the potential to generate significant efficiencies in terms of function evaluations, see [HGD20b] and Section 4.2.2.3. In our grammar the choice of whether (Yes) or not (No) to employ a stopping condition is included as a component, represented by node `<Stopping>`.

4.4.3 Tree-based representation and evolutionary operators

The evolutionary GP process begins with the random generation of a population of heuristics, constructed following the grammar production rules in Figure 4.2. Populating subsequent generations requires the selection, combination and mutation of heuristics.

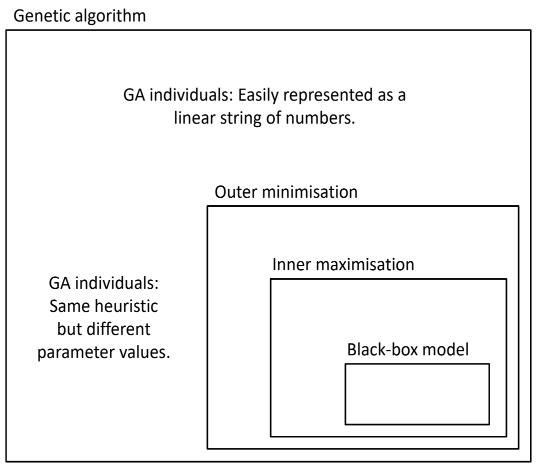
Fitness-based selection can be undertaken as in any standard evolutionary process. Here each heuristic in the population, in each generation, is applied to a single test problem or group of problems, as appropriate. A heuristic is run on any single problem multiple times to generate a sample. For each heuristic applied to each problem, the mean of the samples is used as a fitness measure. If only a single problem is under consideration the fitness across the population of heuristics can be determined directly from a comparison of the means. If multiple problems are considered, means must be calculated for each heuristic across multiple problems. The description of how these means are combined into a single fitness measure for each heuristic is given in the experimental analysis Section 4.5.

The calculated fitnesses are used in tournament selections to identify two parent heuristics per each individual in the following GP generation, see e.g. [ES12]. A number of elite, unchanged, heuristics are also retained from generation to generation.

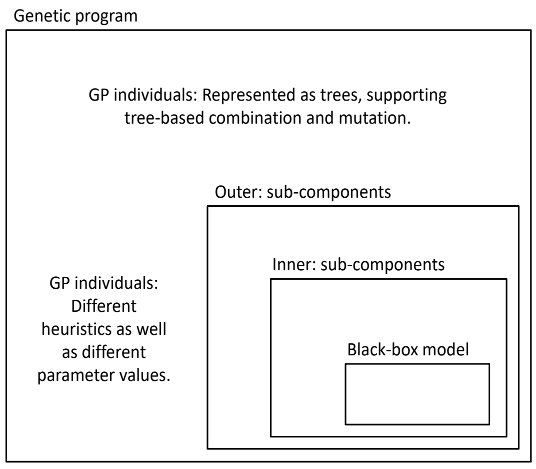
For combination and mutation operations, less standard operators may be required. Consider, for example, the differences between the use of a GA to tune the parameters for a specific heuristic compared to the GP evolution of different heuristics, illustrated in Figure 4.4. All computational evolutionary processes require that an individual object (e.g. a heuristic) has a representative form for the evolution, and in particular combination and mutation operations, to be performed on. In the case of a GA tuning, the solution space consists of the parameters for a single heuristic, which can be represented as a simple linear string of values. So standard GA combination and mutation processes can be employed, see e.g. [Tal09]. Whereas in the GP, each heuristic may comprise different sets of sub-components, complicating a linear representation. For example two such strings would likely be of different lengths, with 'corresponding' sections representing different sub-components and so different parameters. Combining and mutating these strings would introduce difficulties.

Fortunately GP offers an alternative heuristic representation, a tree. This is a common representation for computer programs and algorithms [PDCL05, MHW^{+}10, Noh11, MP17], and lends itself to standard tree-based GP operators. The CFG-GP approach we employ uses this representative form for a heuristic generated by our grammar, Figures 4.2 and 4.3, with standard random tree-based combination and mutation operators [MHW^{+}10, MP16].

Consider the high-level heuristic tree representation in Figure 4.3 with the addition of 'cut' points, Figure 4.5. Any two trees generated by our grammar have this overall structure, so the cut points will apply to all of our heuristics. Any two parent heuristics


Genetic algorithm
GA individuals: Easily represented as a linear string of numbers.
Outer minimisation
Inner maximisation
GA individuals: Same heuristic but different parameter values.
Black-box model

(a) GA parameter tuning


Genetic program
GP individuals: Represented as trees, supporting tree-based combination and mutation.
Outer: sub-components
Inner: sub-components
GP individuals: Different heuristics as well as different parameter values.
Black-box model

(b) Tree-based GP

Figure 4.4: GA and GP applied to a metaheuristic for robust problems, consisting of an outer minimisation search and inner maximisation search operating on a black-box model.

fitness-selected in the GP process, along with one randomly selected cut point, can be combined by merging the branches below the cut in parent tree 1 with the branches above the cut in parent tree 2. The resulting tree is an executable heuristic. This is the combination operation used here.

A newly combined heuristic, represented by a single tree, can be mutated by randomly selecting another cut point. Below the cut point completely new branches can be randomly generated by following the grammar in Figure 4.2, whilst retaining the existing branches from above the cut. This is the mutation operation used here, in conjunction with sampling against a probability of mutation to determine whether to mutate. Thus the requirement for selection, combination and mutation operators applicable directly to the heuristics generated by combining sub-components following our grammar, has been fulfilled.

Note that the generation of the heuristic from the corresponding tree is achieved simply by reading off the sub-components and associated parameter values from the terminal nodes (leaves) at the ends of each branch of the tree.

A final point should be made about the benefits of the CFG-GP approach. In a GP process it is not an absolute requirement to always generate executable algorithms, e.g. a fitness value of zero could be assigned to non-executable algorithms. However it can be appreciated that there is a considerable likelihood of generating non-executable algorithms when randomly combing sub-algorithms. Not only is this very inefficient but it could result in any executable algorithm, whether effective or not, being deemed relatively fit and therefore propagating across many generations. A CFG-GP approach avoids such pitfalls, [MHW$^+$10, MP16].

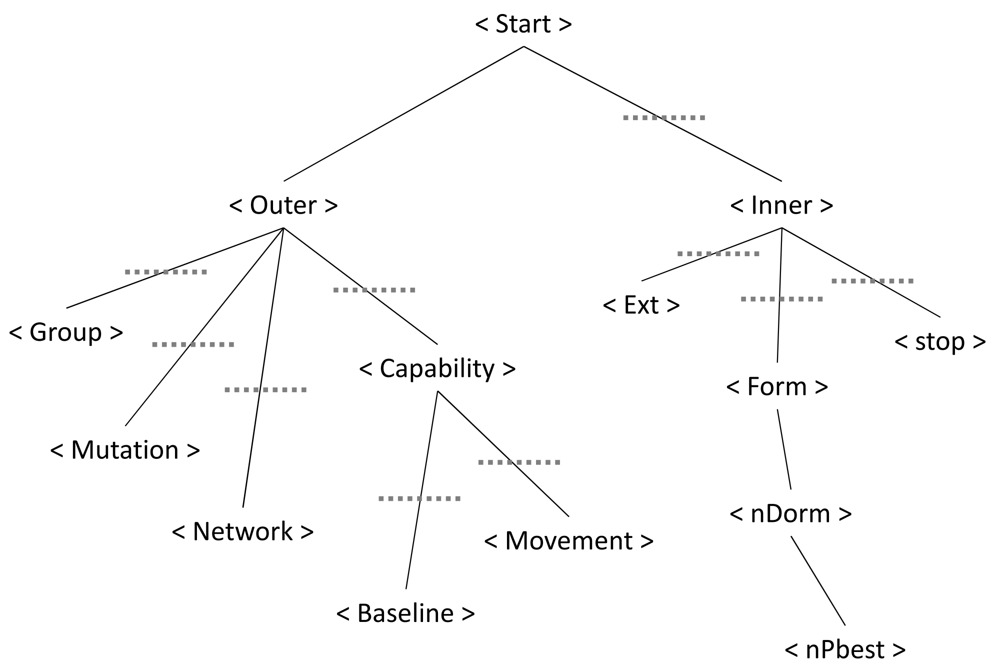


Figure 4.5: High level tree-based representation of the heuristic generated by the grammar production rules in Figure 4.2, with cut points for combination and mutation operations.

4.5 Computational experiments

4.5.1 Experimental set up

The experimental analysis employs 10 established multi-dimensional robust test problems. The problems are listed in Table 4.1 along with the feasible regions and Γ-radius uncertainty values used. Problem formulations and 2D representations are provided in Appendix 4.7.1. In our experiments 30D and 100D versions of these problems are considered.

Name	$\mathcal{X}$	Γ
Rastrigin	$[14.88, 25.12]^n$	0.5
Multipeak F1	$[-5, -4]^n$	0.0625
Multipeak F2	$[10, 20]^n$	0.5
Branke's Multipeak	$[-7, -3]^n$	0.5
Pickelhaube	$[-40, -20]^n$	1
Heaviside Sphere	$[-30, -10]^n$	1
Sawtooth	$[-6, -4]^n$	0.2
Ackley	$[17.232, 82.768]^n$	3
Sphere	$[15, 25]^n$	1
Rosenbrock	$[7.952, 12.048]^n$	0.25

Table 4.1: Test functions.

A single GP run applies each heuristic to a test function or functions, in order to determine fitness and inform the evolutionary process. Here 22 GP runs are considered, once for each test problem individually (individual cases) and once for a combined run where each heuristic is applied to all 10 problems (general case). That is there are 10 individual case GP runs. Within each individual case GP run all heuristics are applied to the same single test problem. There is also one general case GP run, where all heuristics are applied to all 10 test problems. This is repeated for 30D and 100D. A budget of 2,000 function evaluations is assumed in each heuristic run.

In the GP runs, when a heuristic is applied to a problem this is repeated 20 times to generate a mean. For the individual case runs this is taken as the fitness of the heuristic. In a general case run the 10 separate means for each heuristic are used to determine 10 separate fitness rankings. For each heuristic the 10 rankings are averaged to give a combined initial ranking. This ranking is then refined using an elimination process. The worst performing heuristic is ranked lowest and removed. For the remaining heuristics the 10 rankings and combined ranking are recalculated, the new lowest performing heuristic is ranked second lowest overall and removed. This repeats until all heuristics have been ranked.

On completion of a GP run the best heuristic in the final population is accepted. To properly assess its performance 200 sample runs of the heuristic are undertaken, applied to the problem or problems on which it has been evolved. Each run generates an estimate of the location of the robust global optimum for the problem(s) at hand. The corresponding robust value at each global optimum location is re-estimated in a post-processing stage, as the worst value identified by randomly sampling a million points in the Γ-uncertainty neighbourhood of the optimum.

Algorithms are written in Java. For all d.d. calculations the solution of (Soc) includes a call to the IBM ILOG CPLEX Optimization Studio V12.6.3 software.

We now report two analyses. The first, Section 4.5.2, considers the quality of the best solutions (heuristics) found in the GP runs. The second, Section 4.5.3, assesses the structure (component breakdowns) of the heuristics generated in the GP runs, against heuristic performance.

4.5.2 Results for the best performing heuristics

Mean estimates of the optimum robust values for the best performing heuristics, from the 200 sample runs and following the post-processing stage described in Section 4.5.1, are shown in Table 4.2. Corresponding box plots are shown in Figures 4.6 to 4.9. Individual case results are for the best heuristic evolved for a given test problem, then applied to that problem. General case results are for the best general case heuristic at 30D as applied to all 10 test problems, and the best at 100D applied to all 10 problems. The full description of the component breakdowns and parameter values for these best performing heuristics are given in Appendix 4.7.2.

	Individual 30D		Individual 100D		General 30D		General 100D	
	GGGP	Comp	GGGP	Comp	GGGP	Comp	GGGP	Comp
Rastrigin	**154.93**	226.57	**416.35**	648.67	**216.80**	226.57	**615.40**	648.67
Multipeak F1	**-0.64**	-0.63	**-0.64**	-0.58	-0.58	**-0.63**	**-0.62**	-0.58
Multipeak F2	**-0.62**	-0.51	**-0.54**	-0.49	**-0.56**	-0.51	**-0.51**	-0.49
Branke's	0.49	**0.47**	**0.55**	0.56	0.57	**0.47**	0.74	**0.56**
Pickelhaube	**0.43**	**0.44**	**1.04**	1.63	**0.42**	0.44	**1.11**	1.77
Heaviside	**1.03**	1.03	**1.36**	3.32	**1.03**	1.06	**1.57**	3.55
Sawtooth	**0.30**	0.35	**0.25**	0.33	**0.34**	0.35	**0.27**	0.42
Ackley	**5.71**	6.78	**10.33**	14.46	**5.91**	6.78	**10.30**	17.65
Sphere	**1.68**	2.86	**15.60**	36.11	**2.62**	5.30	**19.05**	36.11
Rosenbrock	**55.23**	89.24	**311.96**	1288.00	**57.38**	104.00	**375.01**	1288.00

Table 4.2: Mean estimates of the optimum robust values for the best performing heuristics, due to 200 sample runs and using a budget of 2,000 functions evaluations. Comparators are taken from [HGD20b] and use a budget of 5,000 functions evaluations. Best results are shown in bold.

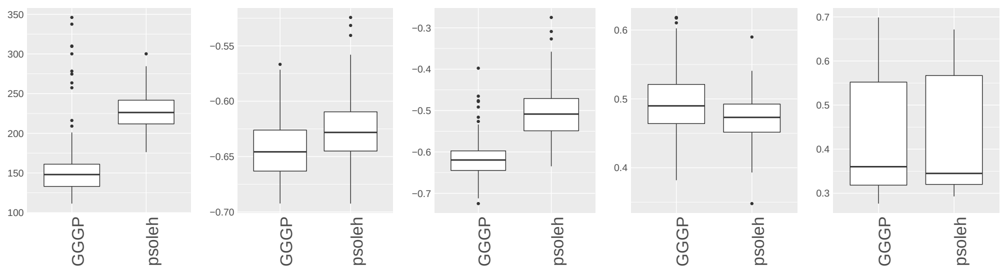
Comparator results taken from [HGD20b] are also shown. There several heuristics were analysed, with each parameter-fitted to 4 of the 10 test problems used here, separately for 30D and 100D. The budget was 5,000 function evaluations. For the individual cases the comparator results shown are due the best performing specific heuristic for an individual problem. For the general case the comparator results are for the best performing heuristic overall in [HGD20b], as applied to all 10 problems.

As the individual case comparators were not tuned on specific problems, comparisons with our individual results should be considered indicative. For the general case a direct comparison is reasonable. Comparisons should be interpreted in the context of the use of a budget of 2,000 function evaluations here. Labels on the box plots, Figures 4.6 to 4.9, give the specific comparator heuristic responsible for each set of results.

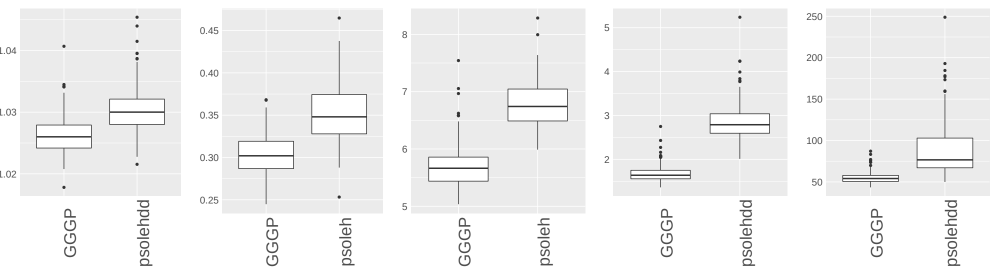
In Table 4.2 values in bold indicate results which are best or statistically equivalent to the best, based on Wilcoxon rank-sum tests with 95% confidence.

At 30D in 8 of the individual cases our GP analysis produces the best heuristic, with one worse than the comparator and one statistically equivalent. For the general case the GP again produces the best heuristic for 8 problems, with the comparator best for 2. In view of our much reduced budget this shows a significantly improved performance for the general case. The individual case comparisons also indicate a good performance.

At 100D the performance of the new heuristics is even better. For all of the individual cases the GP produces the best results. In a number of instances we see substantial improvements, which is encouraging. For the general case the new heuristic is best for 9 problems and worse for one. Again for several problems the new results show significant improvements. In view of the reduced budget this is a strong performance.

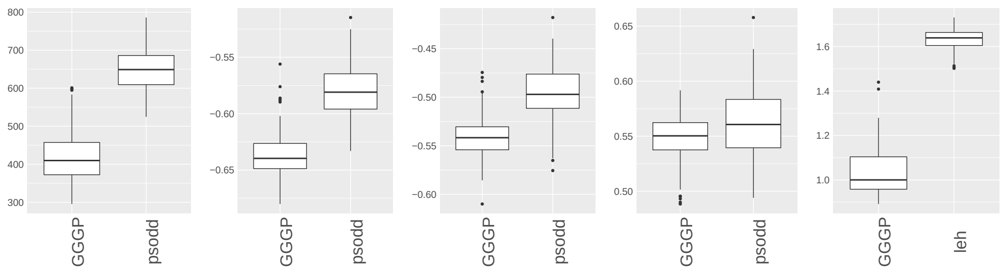


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube

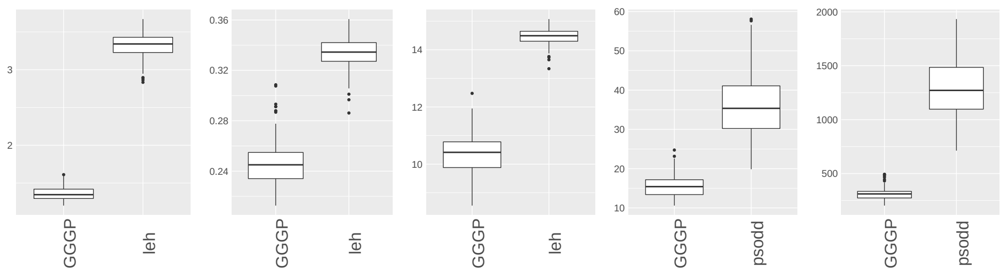


(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.6: 30D individual bests box plots. 200 sample runs with a budget of 2,000 function evaluations. The comparators are taken from [HGD20b], where the budget was 5,000 evaluations.

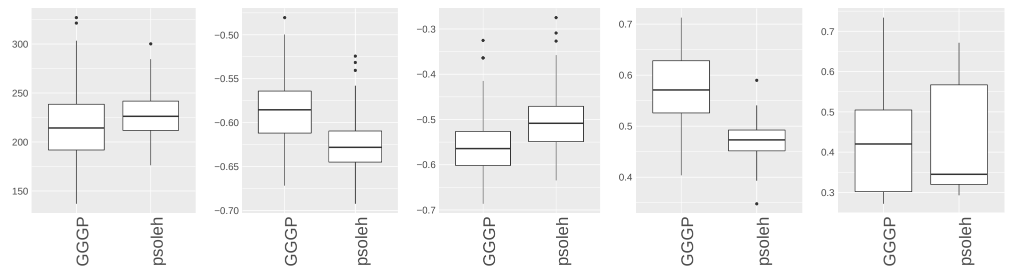


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube

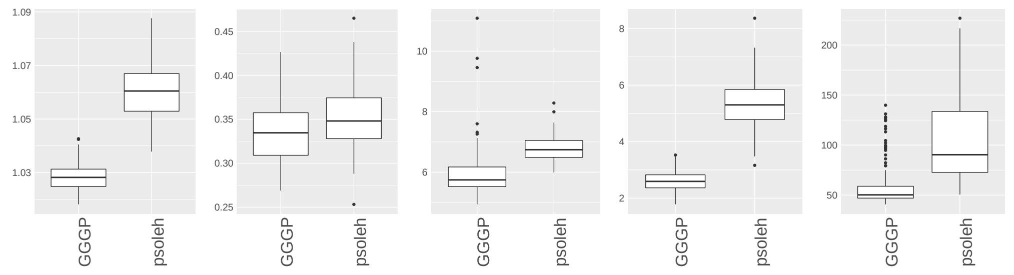


(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.7: 100D individual bests box plots. 200 sample runs with a budget of 2,000 function evaluations. The comparators are taken from [HGD20b], where the budget was 5,000 evaluations.

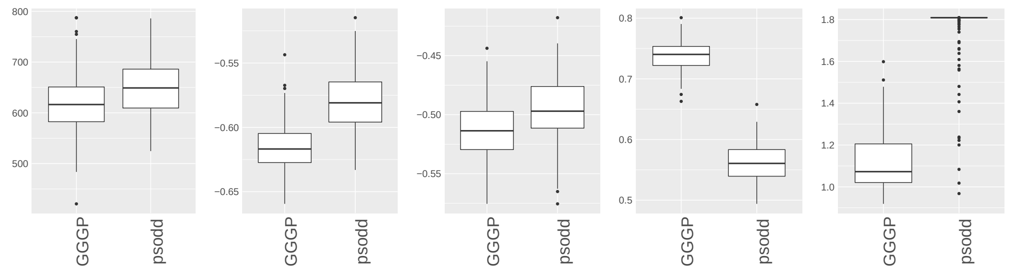


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube

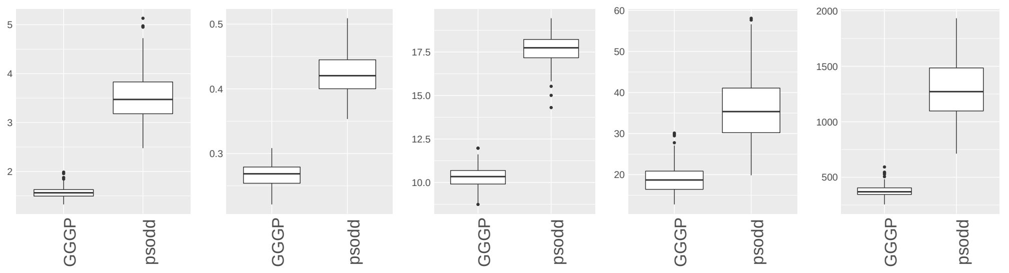


(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.8: 30D best general box plots. 200 sample runs with a budget of 2,000 function evaluations. The comparators are taken from [HGD20b], where the budget was 5,000 evaluations.



(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube



(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.9: 100D best general box plots. 200 sample runs with a budget of 2,000 function evaluations. The comparators are taken from [HGD20b], where the budget was 5,000 evaluations.

4.5.3 Component analysis

4.5.3.1 Analysis of best performing heuristics

The component breakdown for the best heuristic generated in each of the 22 GP runs are shown in Tables 4.3 and 4.4. The results discussed in Section 4.5.2 are generated by these heuristics. We first consider this snapshot of the components associated with the very best performing heuristics, and then move on to consider the component breakdowns across all heuristics from all GP runs, against heuristic performance.

From Tables 4.3 and 4.4 it can be observed that an inner maximisation using random sampling is much preferred, with only 2 heuristics employing an alternative, PSO. Furthermore only a small number of points are typically sampled, with 14 heuristics using 5 points or less in the inner maximisation. In all cases a particle level stopping condition is used.

For the movement formulations, a baseline Inertia velocity formulation is preferred by 20 heuristics. An extended capability is used by all heuristics, with 16 using the full +DD+LEH capability. In all heuristics where dormancy and relocation are used (including +LEH), relocation using the largest empty hypersphere is selected over random relocation. Where a descent direction vector is used (including +DD), a unit vector form of $\boldsymbol{r}_3$ is employed in all but 3 heuristics, rather than a randomised vector.

The best heuristics typically employ small swarm sizes, with 17 using less than 10 particles. Of the 4 network topologies appearing in Tables 4.3 and 4.4, 14 heuristics use Global, with Hierarchical, von Neumann and Ring also represented. Where dormancy is relevant the use of existing information to inform it is preferred in 14 heuristics. The use of existing information to update a robust value on completion of an inner search is preferred 19 times. Some form of PSO level mutation is employed 12 times, 7 of which are by sampling from a Gaussian distribution.

Beyond this narrow snapshot of the component breakdowns of the very best performing search algorithms, an assessment of the forms of component included across the large numbers of heuristics generated by our GP runs will give some indication of how each alternative impacts heuristic performance. The alternative forms that a component may take are given by the grammar in Figure 4.2. For a given component the levels of representation of each alternative form across all heuristics generated in the GP runs, is driven by evolutionary processes and so will indicate some preference. At a component level Table 4.5 gives the proportions of each alternative form separately for 30D and 100D, from all heuristics generated here. For a given dimension the heuristics due to all individual cases and the general case are taken in total.

In Table 4.5 results for the top third best performing heuristics are also shown. This is a high level indication of the impact of each alternative form on heuristic performance. In Figures 4.10 to 4.27 we expand on this information for selected components. Each plot relates to all heuristics generated in a single GP run. By arranging all heuristics generated in a run in order of best-to-worst fitness, an assessment of the heuristic component breakdowns with fitness is made, using a decile scale of heuristic performance (x axis). Within each decile of the fitness-sorted heuristics, for each component, the proportion of each alternative choice for that component is calculated. For a given component and a given test problem case, each line on the plot represents one of the alternatives available for that component. A line is generated from the 10 decile point values, representing the proportion of heuristics within that decile which employ that alternative (y-axis). Within any given decile the plotted values add to 100% as they refer to the proportion of heuristics within that decile.

For example consider Figure 4.10 comprising 10 individual test problem plots at 30D, for the form of inner maximisation. There are 3 alternatives for the inner maximisation: random sampling (red), particle swarm optimisation (green), or genetic algorithm (blue). For the Rastrigin problem, of all of the heuristics within the first decile, that is the top 10% performing heuristics, 97% employ random sampling whilst 2% and 1% employ PSO and GA respectively. These are the values plotted at the 0.1 position on the x-axis. Within the next decile at the 0.2 position on the x-axis (the 10% – 20% range of best performing heuristics), the values are 75%, 7% and 18% for random sampling, PSO and GA respectively.

These plots indicate how component breakdowns relate to heuristic performance. In addition the relative areas under the lines indicate the total proportion of each component category across all heuristics in a single GP run. For example in the Rastrigin plot in Figure 4.10 the proportions of each alternative for the form of inner maximisation is 48%, 24% and 28% for random sampling, PSO and GA respectively. In Sections 4.5.3.2 to 4.5.3.11 we consider the main components individually.

	Group	inner	Movement	Network	inExt	stop	nDorm	nPBest	Mutation	Baseline	Relocate	r3
Rastrigin	2	Random	rPSO+DD+LEH	Global	48	Yes	Yes	Yes	None	Inertia	LEH	Unity
Multipeak F1	21	Random	rPSO+DD+LEH	Global	4	Yes	No	Yes	Uniform	Inertia	LEH	Unity
Multipeak F2	8	Random	rPSO+DD+LEH	Hierarchical	5	Yes	Yes	Yes	Uniform	Inertia	LEH	Unity
Branke's	9	Random	rPSO+DD	Hierarchical	4	Yes	na	Yes	Uniform	Inertia	na	Random
Pickelhaube	3	Random	rPSO+DD+LEH	Global	4	Yes	Yes	Yes	None	Constriction	LEH	Unity
Heaviside	3	Random	rPSO+DD	Global	7	Yes	na	Yes	None	Inertia	na	Unity
Sawtooth	10	Random	rPSO+LEH	Global	6	Yes	No	Yes	Gaussian	Inertia	LEH	na
Ackley	3	Random	rPSO+DD+LEH	Global	4	Yes	Yes	Yes	None	Inertia	LEH	Unity
Sphere	2	Random	rPSO+DD	Hierarchical	15	Yes	na	Yes	None	Inertia	na	Unity
Rosenbrock	9	Random	rPSO+DD+LEH	Global	6	Yes	Yes	No	None	Constriction	LEH	Unity
General	4	Random	rPSO+DD	Ring	4	Yes	na	Yes	Gaussian	Inertia	na	Unity

Table 4.3: 30D components of best heuristics.

	Group	inner	Movement	Network	inExt	stop	nDorm	nPBest	Mutation	Baseline	Relocate	r3
Rastrigin	4	Random	rPSO+DD+LEH	Global	9	Yes	Yes	Yes	Gaussian	Constriction	LEH	Unity
Multipeak F1	8	Random	rPSO+DD+LEH	Global	4	Yes	No	Yes	Uniform	Inertia	LEH	Unity
Multipeak F2	8	Random	rPSO+DD+LEH	von Neumann	5	Yes	Yes	Yes	Gaussian	Inertia	LEH	Unity
Branke's	15	PSO	rPSO+LEH	Hierarchical	4	Yes	No	No	Uniform	Inertia	LEH	na
Pickelhaube	9	Random	rPSO+DD+LEH	Hierarchical	4	Yes	Yes	Yes	None	Inertia	LEH	Unity
Heaviside	8	Random	rPSO+DD+LEH	Global	4	Yes	Yes	No	Gaussian	Inertia	LEH	Unity
Sawtooth	21	Random	rPSO+DD+LEH	von Neumann	4	Yes	Yes	Yes	Gaussian	Inertia	LEH	Unity
Ackley	11	PSO	rPSO+DD+LEH	Global	4	Yes	Yes	Yes	None	Inertia	LEH	Random
Sphere	5	Random	rPSO+DD+LEH	Global	8	Yes	Yes	Yes	None	Inertia	LEH	Unity
Rosenbrock	5	Random	rPSO+DD+LEH	Global	8	Yes	Yes	Yes	Gaussian	Inertia	LEH	Random
General	8	Random	rPSO+DD+LEH	Global	4	Yes	Yes	Yes	None	Inertia	LEH	Unity

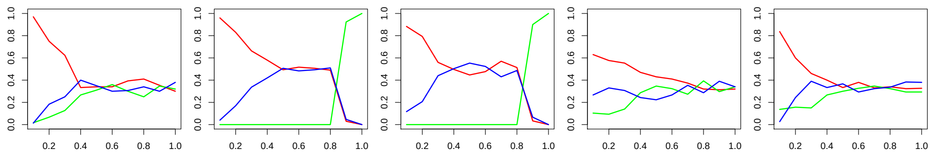
Table 4.4: 100D components of best heuristics.

Component		30D all	30D top	100D all	100D top
Form of inner search	**Random**	46.6%	67.5%	42.5%	55.3%
	PSO	25.2%	12.8%	29.8%	25.5%
	GA	28.1%	19.8%	27.7%	19.2%
Extent of inner search	**[2-10]**	49.9%	83.0%	50.3%	84.9%
	[11-20]	21.1%	12.8%	20.3%	10.3%
	[21-30]	11.2%	2.4%	11.5%	3.5%
	[31-40]	5.3%	0.6%	5.7%	0.8%
	>40	12.5%	1.3%	12.2%	0.6%
Form of baseline rPSO formula	**Constriction**	36.2%	17.0%	35.8%	19.5%
	Inertia	63.8%	83.0%	64.2%	80.5%
Form of movement	**rPSO**	12.1%	3.1%	11.4%	1.6%
	+DD	25.0%	25.2%	15.0%	7.8%
	+LEH	25.8%	25.0%	32.2%	36.3%
	+DD+LEH	37.1%	46.8%	41.4%	54.3%
Form of network	**Global**	24.5%	41.2%	25.6%	43.0%
	Focal	9.8%	3.1%	9.9%	3.2%
	Ring (size=2)	11.3%	6.9%	11.1%	6.4%
	von Neumann	13.6%	12.8%	14.3%	12.9%
	Clan	13.2%	10.9%	13.6%	13.5%
	Cluster	12.1%	8.8%	11.5%	7.9%
	Hierarchy	15.5%	16.5%	14.1%	13.1%
Group (swarm) size	**[2-10]**	25.6%	43.2%	22.4%	33.9%
	[11-20]	19.4%	17.0%	21.3%	22.1%
	[21-30]	20.2%	18.0%	20.3%	18.1%
	[31-40]	17.6%	11.9%	18.2%	13.8%
	>40	17.2%	10.0%	17.8%	12.1%
Inclusion of stopping condition	**No**	36.8%	16.2%	40.6%	26.9%
	Yes	63.2%	83.8%	59.4%	73.1%
Use of existing info. for dormancy	**No**	30.4%	32.3%	32.3%	33.4%
	Yes	32.6%	39.4%	41.3%	57.3%
	Not applicable	37.0%	28.3%	26.4%	9.4%
Use of existing info. for personal best	**No**	46.3%	39.9%	44.6%	36.5%
	Yes	53.7%	60.1%	55.4%	63.5%
Form of mutation	**None**	37.2%	42.4%	39.1%	48.0%
	Random	31.0%	27.2%	31.9%	29.9%
	Gaussian	31.8%	30.4%	29.0%	22.1%
Form of relocation due to dormancy	**LEH**	55.5%	71.7%	65.7%	90.6%
	Random	7.5%	0.0%	7.9%	0.0%
	Not applicable	37.0%	28.3%	26.4%	9.4%
Form of r3 vector	**Random**	27.3%	27.9%	29.6%	32.3%
	Unity	34.8%	44.0%	26.8%	29.8%
	Not applicable	37.9%	28.1%	43.6%	37.9%

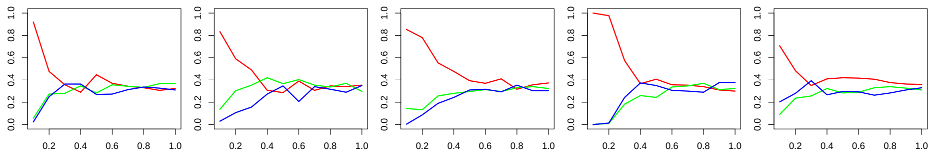
Table 4.5: Proportions of component make ups over all heuristics. Here 'top' refers to the top one third of heuristics when sorted best to worst.

4.5.3.2 Form and extent of inner maximisation

From Table 4.5 and Figures 4.10 to 4.12 it can be seen that the most used form of inner maximisation search is random sampling. For the 30D individual cases random sampling is the most commonly associated choice with the best performing heuristics, dominating for most problems. At 100D random sampling is again most typically associated with the best heuristics, although it is much less dominant, with PSO the most used form of inner search in the best performing heuristics for the Branke and Heaviside problems. For the general cases random sampling dominates the best performing heuristics.

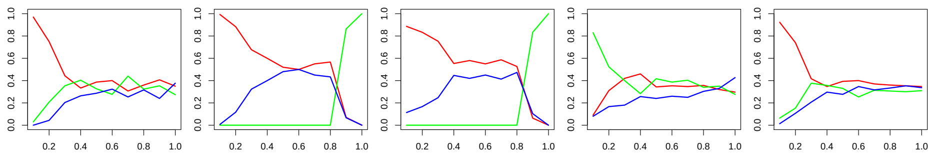


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube

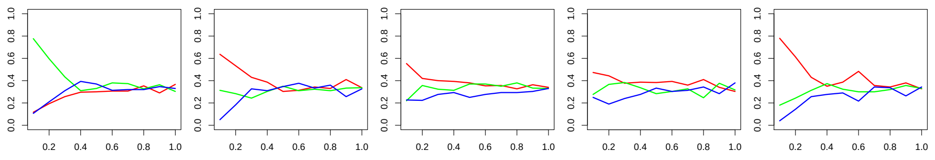


(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.10: Component – decile breakdowns for the form of inner search, across all GGGP heuristics at 30D. Components: Random (red), PSO (green), GA (blue).

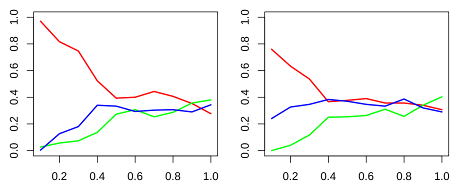


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube



(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

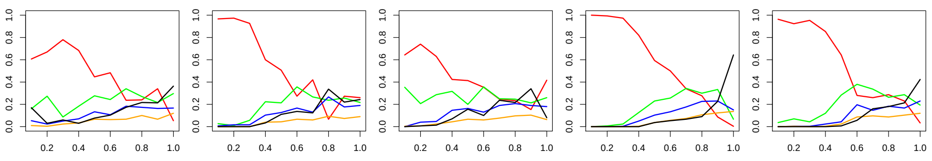
Figure 4.11: Component – decile breakdowns for the form of inner search, across all GGGP heuristics at 100D. Components: Random (red), PSO (green), GA (blue).



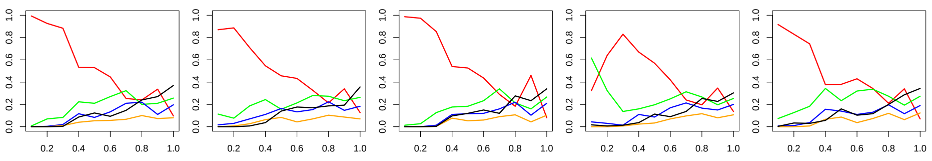
(a) 30D (b) 100D

Figure 4.12: Component – decile breakdowns for the form of inner search, across all GGGP heuristics at 30D and 100D for the general heuristics. Components: Random (red), PSO (green), GA (blue).

In terms of the number of points evaluated in an inner maximisation, Table 4.5 and Figures 4.13 to 4.15 show a clear dominance for low numbers, primarily in the range 2-10 (red). This is both for all heuristics generated and those performing best. Additional analysis shows that for the best performing third of heuristics, the proportions that employ random sampling using a number of points in the 2-10 range is 56% and 49% for 30D and 100D respectively.

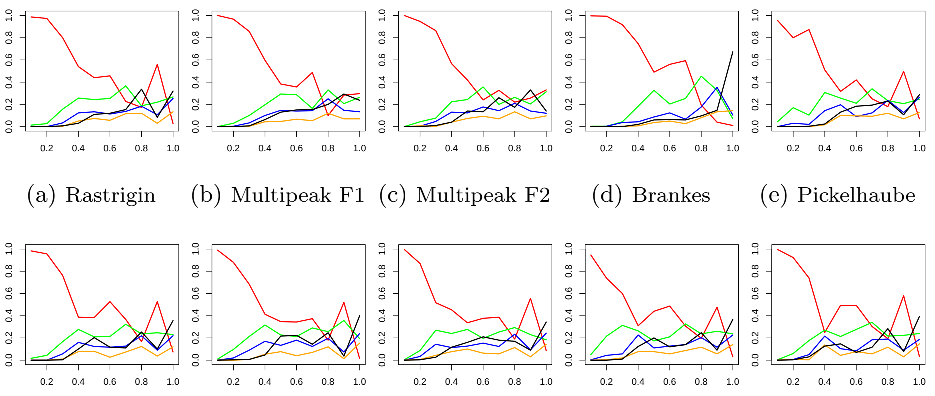


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube



(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

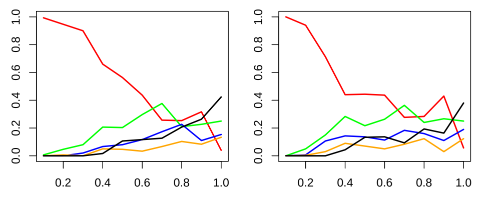
Figure 4.13: Component – decile breakdowns for the extent (size) of the inner maximisation search, across all GGGP heuristics at 30D. Components: $[2-10]$ (red), $[11-20]$ (green), $[21-30]$ (blue), $[31-40]$ (orange), > 40 (black).



(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube

(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.14: Component – decile breakdowns for the extent (size) of the inner maximisation search, across all GGGP heuristics at 100D. Components: [2 – 10] (red), [11 – 20] (green), [21 – 30] (blue), [31 – 40] (orange), > 40 (black).



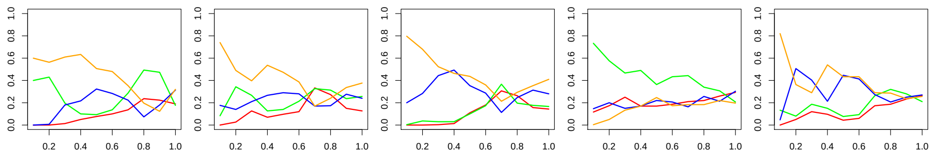
(a) 30D (b) 100D

Figure 4.15: Component – decile breakdowns for the extent (size) of the inner maximisation search, across all GGGP heuristics at 30 D and 100D for the general heuristics. Components: [2 – 10] (red), [11 – 20] (green), [21 – 30] (blue), [31 – 40] (orange), > 40 (black).

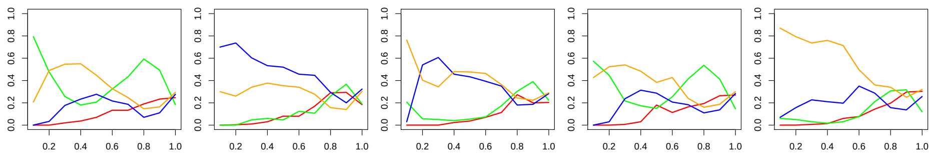
4.5.3.3 Baseline and extended movement capabilities

Table 4.5 shows that the inertia formulation of the baseline particle velocity equation appears in 64% of all heuristics for both 30D and 100D, whilst for the best performing third it dominates, appearing in over 80% of heuristics. A decile level analysis confirms this dominance in the best performing heuristics.

However for the extended form of particle level movement, things are less clear. From Table 4.5 it can be seen that the most used form of extended capability includes both descent direction and LEH dormancy-relocation (+DD+LEH) for both 30D and 100D. For both dimensions this increases in the top third performing heuristics. Nevertheless both the +LEH and +DD individual capabilities are also well represented. Figures 4.16 to 4.18 show the decile analysis for the extended movement capability: no additional capability (red: rPSO), +DD (green), +LEH (blue), or +DD+LEH (orange). The plots are in accord with the high level results, with +DD+LEH most associated with the best performing heuristics but both +DD and +LEH also performing well.

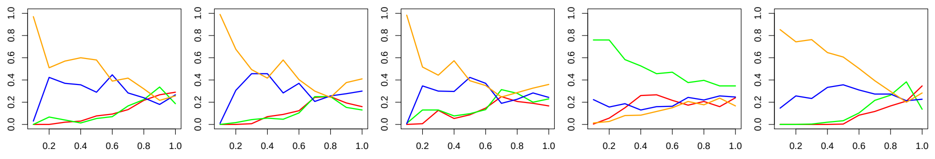


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube

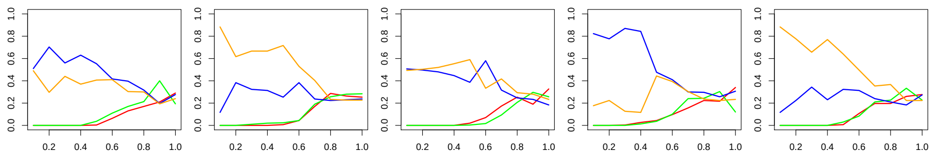


(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.16: Component – decile breakdowns for the form of movement capability, across all GGGP heuristics at 30D. Components: Baseline (red), DD (green), LEH (blue), LEH+DD (orange).

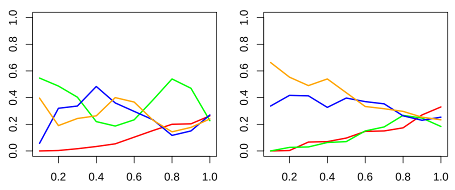


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube



(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.17: Component – decile breakdowns for the form of movement capability, across all GGGP heuristics at 100D. Components: Baseline (red), DD (green), LEH (blue), LEH+DD (orange).



(a) 30D (b) 100D

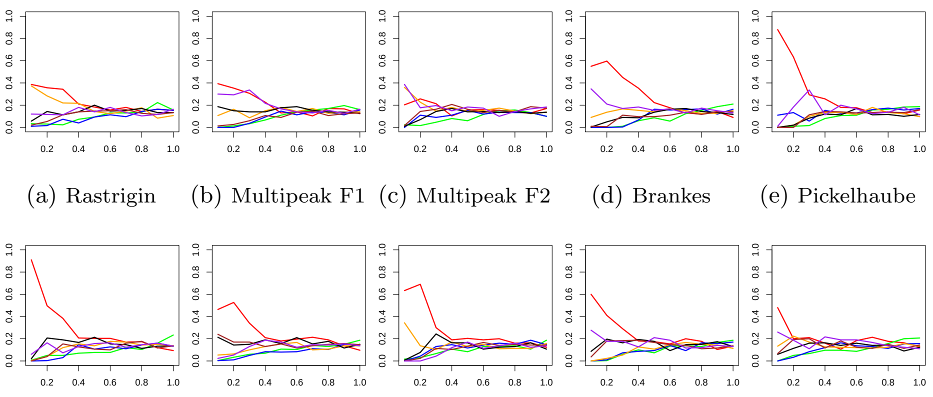
Figure 4.18: Component – decile breakdowns for the form of movement capability, across all GGGP heuristics at 30D and 100D for the general heuristics. Components: Baseline (red), DD (green), LEH (blue), LEH+DD (orange).

4.5.3.4 Network topology

For the form of network for particle information sharing, Table 4.5 shows that a Global (red) network is most preferred for both 30D and 100D, with this increasing in the top third performing heuristics. No other forms of network particularly stand out. This is reflected in the decile plots for the individual case heuristics, Figures 4.19 and 4.20, although both Hierarchical (purple) and von Neumann (orange) networks are also represented in the lowest deciles for a few problems. In the general cases, Figure 4.21, at 30D the Ring (blue) network outperforms Global for the lowest deciles.

4.5.3.5 Group size

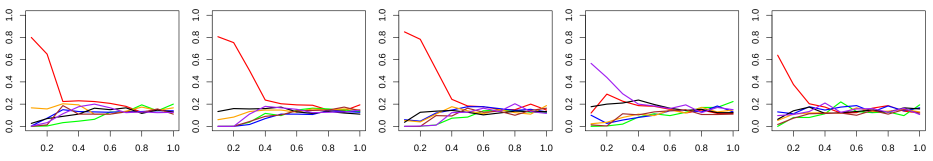
For the group (outer PSO swarm) size, in Table 4.5 all of the different categories are quite well represented considering all heuristics generated in the GP runs. Most common is the lowest range, 2-10 (red), and in the best performing third of heuristics this range stands out somewhat, increasing to 43% and 34% for 30D and 100D respectively. A similar pattern is observed for the individual case heuristics decile analysis in Figures 4.22 and 4.23, although the next group size range, 11-20 (green), is also favoured in the lowest deciles for a few problems and in particular at 100D. For the general case at 100D, Figure 4.24, the lower decile results are well distributed across the group size ranges.



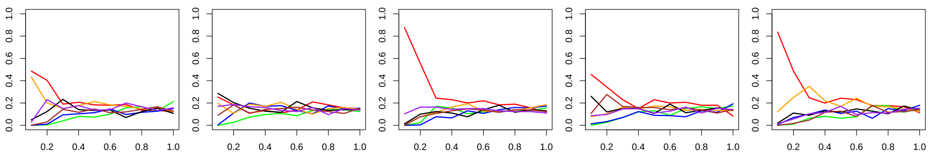
(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube

(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.19: Component – decile breakdowns for the form of network for particle information sharing, across all GGGP heuristics at 30D. Components: Global (red), Focal (green), Ring (size=2) (blue), von Neumann (orange), Clan (black), Cluster (brown), Hierarchy (purple).

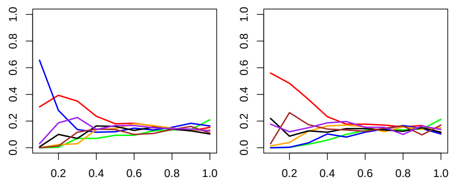


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube



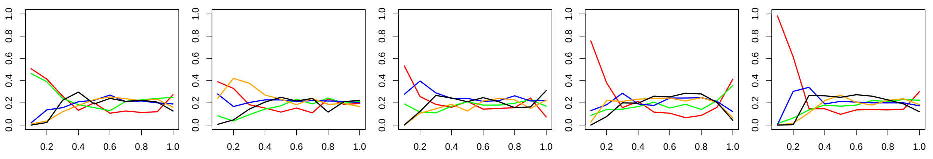
(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.20: Component – decile breakdowns for the form of network for particle information sharing, across all GGGP heuristics at 100D. Components: Global (red), Focal (green), Ring (size=2) (blue), von Neumann (orange), Clan (black), Cluster (brown), Hierarchy (purple).

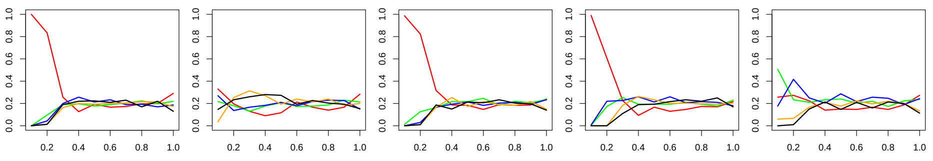


(a) 30D (b) 100D

Figure 4.21: Component – decile breakdowns for the form of network for particle information sharing, across all GGGP heuristics at 30D and 100D for the general heuristics. Components: Global (red), Focal (green), Ring (size=2) (blue), von Neumann (orange), Clan (black), Cluster (brown), Hierarchy (purple).

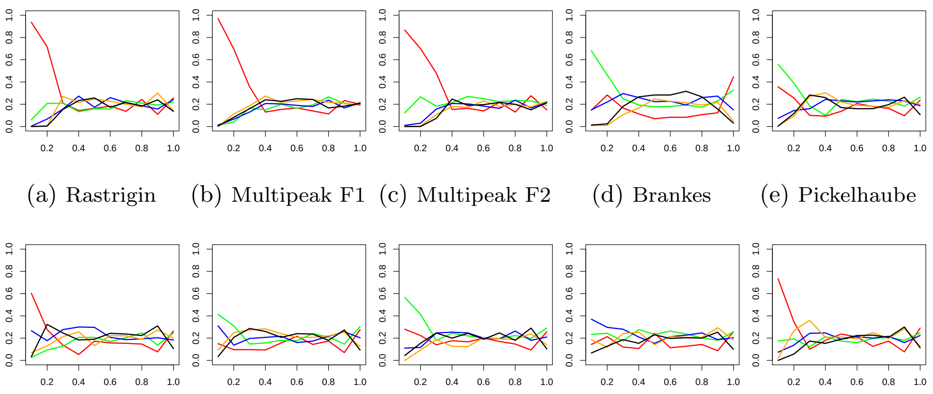


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube



(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

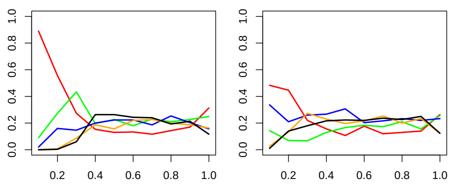
Figure 4.22: Component – decile breakdowns for the group (swarm) size, across all GGGP heuristics at 30D. Components: [2 – 10] (red), [11 – 20] (green), [21 – 30] (blue), [31 – 40] (orange), > 40 (black).



(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube

(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.23: Component − decile breakdowns for the group (swarm) size, across all GGGP heuristics at 100D. Components: [2 − 10] (red), [11 − 20] (green), [21 − 30] (blue), [31 − 40] (orange), > 40 (black).



(a) 30D (b) 100D

Figure 4.24: Component − decile breakdowns for the group (swarm) size, across all GGGP heuristics at 30D and 100D for the general heuristics. Components: [2 − 10] (red), [11 − 20] (green), [21 − 30] (blue), [31 − 40] (orange), > 40 (black).

4.5.3.6 Use of a stopping condition

Considering the inclusion of a stopping condition in the inner maximisation, from Table 4.5 it can be seen that this is preferred in 63% and 59% of all heuristics from the GP runs, for 30D and 100D respectively. In the top third of results this increases to 84% and 73%. A decile level analysis confirms this dominance in the best performing heuristics across all GP runs.

4.5.3.7 Use of neighbourhood information for dormancy

When a heuristic uses the dormancy-relocation capability, (+LEH or +DD+LEH), the assessment of dormancy may (Yes) or may not (No) make use of existing uncertainty neighbourhood information from the history set. Table 4.5 indicates that the use or non-use of this information is quite evenly apportioned, particularly for 30D. These high level results are reflected at the individual case and general case decile level analysis. In a few cases the use of information performs better in the lowest deciles.

4.5.3.8 Use of neighbourhood information for $\boldsymbol{x}_*^j$

For the use of uncertainty neighbourhood information in the history set to update a particle's robust value on completion of the inner maximisation search, the use of such information (Yes) versus non-use (No) is somewhat evenly apportioned – although there is some limited preference for using the information. Again these results are reflected at the more detailed decile level analysis.

4.5.3.9 Particle level mutation

Our grammar includes the ability to mutate a particle's intended position. Analysis of this component is shown in Table 4.5 and Figures 4.25 to 4.27. A choice of no mutation (red), or mutation due to either Uniform (green) or Gaussian (blue) random sampling, is available. In Table 4.5 all three alternatives are well represented, with no mutation performing best, appearing in over 40% of the top third performing heuristics. In the decile analysis, for 30D individual cases preference is quite even, whilst at 100D the non-use of mutation is more preferred at the lowest deciles. In the general cases apportionment is evenly distributed.

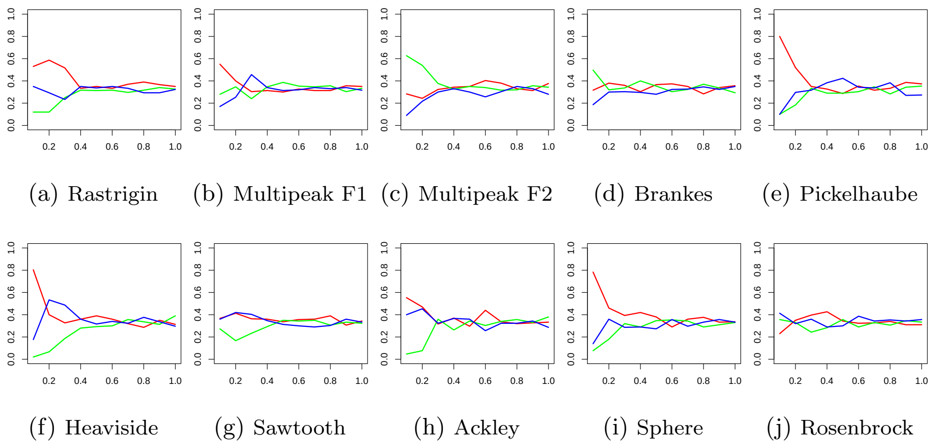
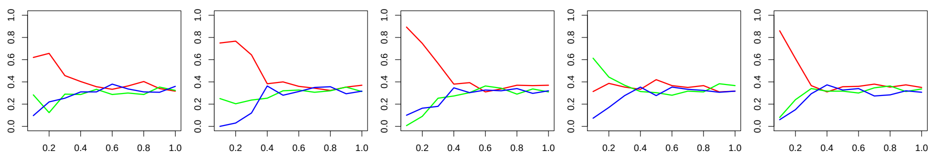
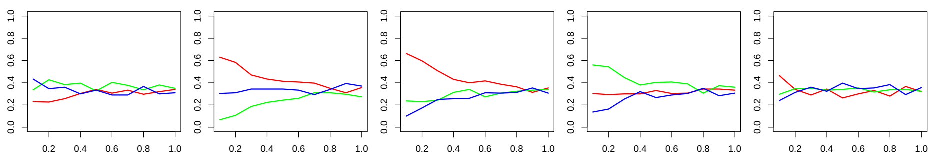


Figure 4.25: Component – decile breakdowns for the form of PSO mutation, across all GGGP heuristics at 30D. Components: None (red), Uniform (green), Gaussian (blue).

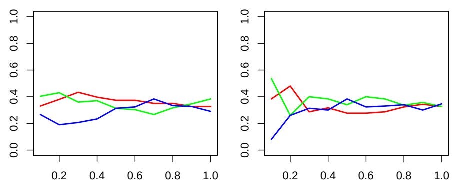


(a) Rastrigin (b) Multipeak F1 (c) Multipeak F2 (d) Brankes (e) Pickelhaube



(f) Heaviside (g) Sawtooth (h) Ackley (i) Sphere (j) Rosenbrock

Figure 4.26: Component – decile breakdowns for the form of PSO mutation, across all GGGP heuristics at 100D. Components: None (red), Uniform (green), Gaussian (blue).



(a) 30D (b) 100D

Figure 4.27: Component – decile breakdowns for the form of PSO mutation, across all GGGP heuristics at 30D and 100D for the general heuristics. Components: None (red), Uniform (green), Gaussian (blue).

4.5.3.10 Extended movement capability: relocation by LEH

If a heuristic employs the dormancy-relocation capability (+LEH or +DD+LEH), there is a choice of how dormant particles are relocated. This is by the calculation of the largest empty hypersphere devoid of poor points, or completely randomly. From Table 4.5 the use of relocation using LEH is seen to dominate when dormancy is employed. At 30D 88% of all heuristics use LEH-relocation, rising to 100% of the top performers. At 100D 89% rises to 100%. This complete dominance is also observed in the decile level analysis.

4.5.3.11 Extended movement capability: descent directions $\boldsymbol{r}_3$ vector

When a heuristic use the descent directions capability (+DD or +DD+LEH), the $\boldsymbol{r}_3$ vector may be generated randomly or set to the unit vector. Table 4.5 indicates a reasonably even use of the randomised or unit $\boldsymbol{r}_3$ vectors. Where d.d. is employed, the use of a unit vector rises from 56% across all heuristics to 61% in the top performers, for 30D. Whereas at 100D this figure remains static at 48%. The decile level analysis reflects these high level patterns. At 30D for several cases the use of a unit vector performs better in the lowest deciles, whilst at 100D the preference is fairly evenly distributed across cases.

4.5.4 Summary of experimental analysis

The analysis of the results due to the best heuristics generated in the GP runs shows a strong performance. For individual case performance the indications against comparator results is encouraging. For the general cases the newly developed heuristics show an improvement over the best comparators, in some cases significant, despite using a budget 60% lower.

For the component level analysis, inner maximisation using random sampling on a small number of points performed best, with a particle level stopping condition strongly preferred. For the outer minimisation the best heuristic performance is (separately) related to a relatively small swarm size, communication using a Global typology, and a particle movement formulation consisting of an inertia based velocity equation plus d.d. and LEH extended capabilities.

In addition to the decile level component analysis reported here, consideration was given to potential correlations between alternatives across different components. No such correlation was observed.

4.6 Conclusions and further work

We have used grammar-guided genetic programming to automatically generate particle swarm based metaheuristics for robust problems, in order to determine improved search algorithms and assess the effectiveness of various algorithmic sub-components. This has involved the generation of a grammar consisting of a number of heuristic building blocks, the design rules for constructing heuristics from these components, and an evolutionary GP process. We have searched a heuristic sub-algorithm space not previously investigated, encompassing specialised robust-focussed capabilities alongside more standard elements such as network topologies and alternatives for the inner maximisation.

Using a suite of 10 test problems at 30D and 100D, the best evolved heuristics were identified at individual and general (all problems simultaneously) test case levels. Using comparators, significant improvements are observed for the best new general heuristics, whilst indicative individual case results are highly promising.

The GP process generates substantial numbers of heuristics, enabling an assessment of algorithmic sub-components against heuristic performance. In the context of a budget of 2,000 function evaluations, this identifies an inner maximisation by random sampling on a small number of points as most effective, including the use of a particle level stopping condition. For the outer minimisation small numbers of particles are preferred, sharing information through a Global topology. Other topologies exhibit some good performance. The preferred particle movement uses a baseline inertia velocity equation plus some largest empty hypersphere [HGW19, HGD20b] and descent directions

[BNT10b, HGD20b] heuristic capabilities. This includes the assessment of particle dormancy and relocation to the centre of the LEH, or the use of a d.d. vector component in the velocity formulation, or both.

There are a number of ways in which this work can be built upon, most obviously in terms of extending the sub-algorithmic space over which the GP operates. Moving away from a PSO structure for all of the heuristics to a more general agent based setting, using other population based metaheuristics, would introduce alternative movement and information sharing capabilities into our grammar for the outer minimisation layer.

As the use of random sampling for the inner maximisation layer has proven effective here, the inclusion in our grammar of some efficient sampling techniques such as the specialised Latin hypercube approach described in [FBG19], would seem appropriate.

The potential efficiencies offered by emulation in either the outer minimisation or inner maximisation layers, warrants investigation. The introduction of emulation based components into the grammar, including sub-elements of specific emulation approaches, could significantly extend the heuristic solution space.

A final consideration might be the use of alternatives to the GGGP approach, to automatically generate heuristics for robust problems.

4.7 Appendix

4.7.1 Test functions

The mathematical descriptions for the 10 test functions used in the experimental testing are given below, with 3D plots of their 2D versions shown in Figure 4.28. All functions are taken from the literature: [Bra98, KEB10, KRD+11, Kru12, JY13].

Rastrigin

$$f(\boldsymbol{x}) = 10n + \sum_{i=1}^{n} \left[(x_i - 20)^2 - 10\cos(2\pi(x_i - 20))\right]$$

The feasible region is the hypercube $x_i \in [14.88, 25.12]$.

MultipeakF1

$$f(\boldsymbol{x}) = -\frac{1}{n}\sum_{i=1}^{n} g(x_i)$$

$$g(x_i) = \begin{cases} e^{-2\ln 2\left(\frac{(x_i+5)-0.1}{0.8}\right)^2} \sqrt{|\sin(5\pi(x_i+5))|} & \text{if } 0.4 < x_i + 5 \leq 0.6 \text{ ,} \\ e^{-2\ln 2\left(\frac{(x_i+5)-0.1}{0.8}\right)^2} \sin^6(5\pi(x_i+5)) & \text{otherwise} \end{cases}$$

The feasible region is the hypercube $x_i \in$ [-5, -4].

MultipeakF2

$$f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} g(x_i)\ ,\ g(x_i) = 2\sin(10e^{-0.2(x_i-10)}(x_i-10))e^{-0.25(x_i-10)}$$

The feasible region is the hypercube $x_i \in$ [10, 20].

Branke's Multipeak

$$f(\boldsymbol{x}) = \max\{c_1, c_2\} - \frac{1}{n}\sum_{i-1}^{n} g(x_i)$$

$$g(x_i) = \begin{cases} c_1\left(1 - \frac{4((x_i+5)+\frac{b_1}{2})^2}{b_1^2}\right) & \text{if } -b_1 \le (x_i+5) < 0\ , \\ c_2 \cdot 16^{\frac{-2|b_2-2(x_i+5)|}{b_2}} & \text{if } 0 \le (x_i+5) \le b_2\ , \\ 0 & \text{otherwise} \end{cases}$$

Here $b_1 = 2$, $b_2 = 2$, $c_1 = 1$, $c_2 = 1.3$.
The feasible region is the hypercube $x_i \in$ [-7, -3].

Pickelhaube

$$f(\boldsymbol{x}) = \frac{5}{5-\sqrt{5}} - \max\{g_0(\boldsymbol{x}), g_{1a}(\boldsymbol{x}), g_{1b}(\boldsymbol{x}), g_2(\boldsymbol{x})\}$$

$$g_0(\boldsymbol{x}) = \frac{1}{10}e^{-\frac{1}{2}\|\boldsymbol{x}+30\|}$$

$$g_{1a}(\boldsymbol{x}) = \frac{5}{5-\sqrt{5}}\left(1 - \sqrt{\frac{\|\boldsymbol{x}+30+5\|}{5\sqrt{n}}}\right)$$

$$g_{1b}(\boldsymbol{x}) = c_1\left(1 - \left(\frac{\|\boldsymbol{x}+30+5\|}{5\sqrt{n}}\right)^4\right)$$

$$g_2(\boldsymbol{x}) = c_2\left(1 - \left(\frac{\|\boldsymbol{x}+30-5\|}{5\sqrt{n}}\right)^{d_2}\right)$$

Here $c_1 = 625/624$, $c_2 = 1.5975$, $d_2 = 2 = 1.1513$.
The feasible region is the hypercube $x_i \in$ [-40, -20].

Heaviside Sphere

$$f(\boldsymbol{x}) = \left(1 - \prod_{i=1}^{n} g(x_i)\right) + \sum_{i=1}^{n} \left(\frac{(x_i + 20)}{10}\right)^2$$

$$g(x_i) = \begin{cases} 0 & \text{if } 0 < (x_i + 20) \ , \\ 1 & \text{otherwise} \end{cases}$$

The feasible region is the hypercube $x_i \in$ [-30, -10].

Sawtooth

$$f(\boldsymbol{x}) = 1 - \frac{1}{n} \sum_{i=1}^{n} g(x_i) \ , \ g(x_i) = \begin{cases} (x_i + 5) + 0.8 & \text{if } -0.8 \leq (x_i + 5) < 0.2 \ , \\ 0 & \text{otherwise} \end{cases}$$

The feasible region is the hypercube $x_i \in$ [-6, -4].

Ackleys

$$f(\boldsymbol{x}) = -20 \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - 50)^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi(x_i - 50))\right) + 20 + \exp(1)$$

The feasible region is the hypercube $x_i \in$ [17.232, 82.768].

Sphere

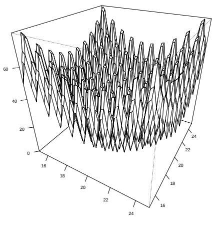
$$f(\boldsymbol{x}) = \sum_{i=1}^{n} (x_i - 20)^2$$

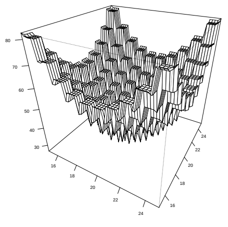
The feasible region is the hypercube $x_i \in$ [15, 25].

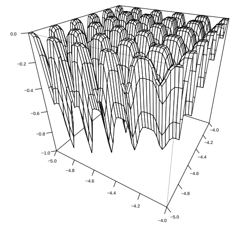
Rosenbrock

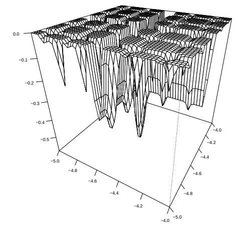
$$f(\boldsymbol{x}) = \sum_{i=1}^{n-1} [100((x_{i+1} - 10) - (x_i - 10)^2)^2 + ((x_i - 10) - 1)^2]$$

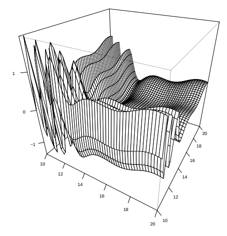
The feasible region is the hypercube $x_i \in$ [7.952, 12.048].

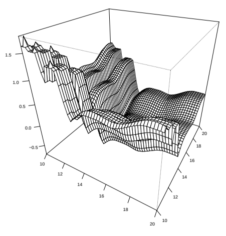

(a) Rastrigin Nom

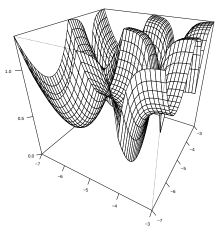

(b) Rastrigin Worst

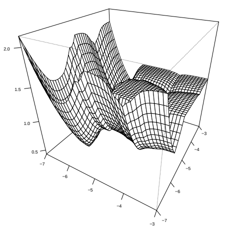

(c) Multipeak F1 Nom

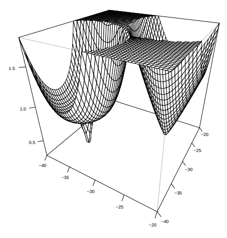

(d) Multipeak F1 Worst

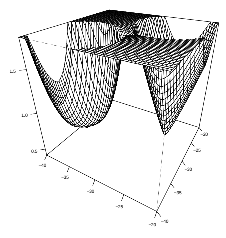

(e) Multipeak F2 Nom

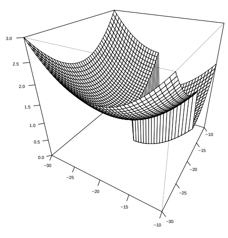

(f) Multipeak F2 Worst

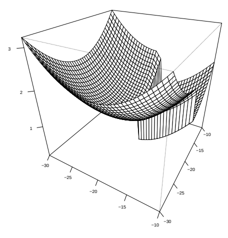

(g) Brankes Multi Nom

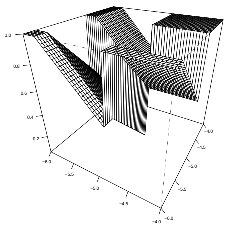

(h) Brankes Multi Worst

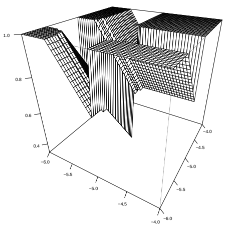

(i) Pickelhaube Nom

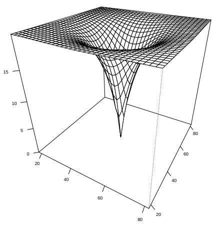

(j) Pickelhaube Worst

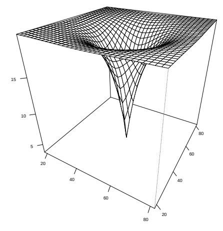

(k) Heaviside S Nom

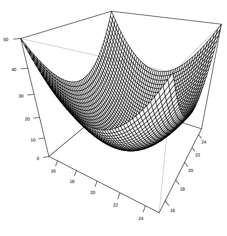

(l) Heaviside S Worst

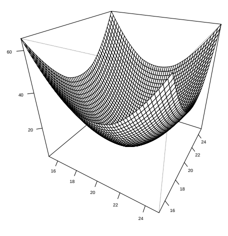

(m) Sawtooth Nom

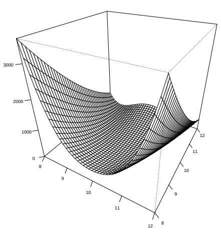

(n) Sawtooth Worst


(o) Ackley Nom


(p) Ackley Worst


(q) Sphere Nom


(r) Sphere Worst


(s) Rosenbrock Nom

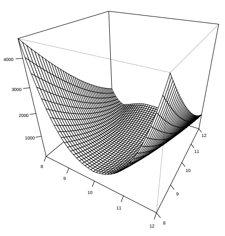

(t) Rosenbrock Worst

Figure 4.28: Plots of 2D versions of the functions used in our experimental testing.

4.7.2 Heuristic parameter-component values

The full list of parameter-component values for the best performing heuristics as reported on in Tables 4.2 to 4.4, are shown below in Tables 4.6 to 4.9. The components are as described in our grammar, Figure 4.2

For the particle-iteration level d.d. calculations used here, the σ value can be reduced a fixed ten times in ten equal steps from it's initial value (σ) to the σ_{limit} value. This applies when the re-calculation of a high cost set is required. A fixed number of steps is employed here to limit processing time. In addition, for each particle at each iteration, a fixed minimum step size is used: (Min step * Γ). See [BNT10b] for a full description of the d.d. algorithm. Also, in the LEH calculation, the mutation 'amount' ('lmutA' in the grammar Figure 4.2) is actually a percentage value which is subsequently multiplied by the dimensional range of the decision variable space $\mathcal{X}$ to give the actual amount by which any value is adjusted due to mutation.

	Rastrigin	Multi F1	Multi F2	Brankes	Pickel	Heavi
Group size	2	21	8	9	3	3
Mutation	None	Uniform	Uniform	Uniform	None	None
Mutation prob	na	0.3810	0.3106	0.1511	na	na
Network	Global	Global	Hierarch	Hierarch	Global	Global
Baseline	Inertia	Inertia	Inertia	Inertia	Constrict	Inertia
C1 (base)	1.6757	1.3441	2.2915	2.3900	2.0545	1.1573
C2 (base)	1.6501	0.8235	1.0337	0.8500	2.1980	0.5565
ω (base)	0.2084	0.4674	0.2552	0.7000	na	0.2579
Movement	DD+LEH	DD+LEH	DD+LEH	DD	DD+LEH	DD
C3 (DD)	4.0812	7.2490	8.6110	4.5671	7.1072	7.6501
σ (DD)	0.3315	0.3673	0.3593	0.3116	0.3970	0.1337
σ_{limit} (DD)	0.0033	0.0063	0.0087	0.0041	0.0037	0.0077
Min step (DD)	0.0793	0.0382	0.0154	0.0728	0.0756	0.0684
r3 (DD)	1	1	1	Random	1	1
Relocation (LEH)	LEH	LEH	LEH	na	LEH	na
GA pop (LEH)	25	20	25	na	25	na
GA gens (LEH)	4	5	4	na	4	na
GA mut prob (LEH)	0.0294	0.4991	0.9189	na	0.1966	na
GA mut perc (LEH)	0.0557	0.0191	0.0892	na	0.0210	na
GA elites (LEH)	1	3	2	na	2	na
GA tour size (LEH)	2	7	3	na	9	na
Dormancy limit (LEH)	2	1	1	na	3	na
Inner extent	48	4	5	4	4	7
Inner max	Random	Random	Random	Random	Random	Random
Use n'hood: dormancy	Yes	No	Yes	Yes	Yes	No
Use n'hood: pBest	Yes	Yes	Yes	Yes	Yes	Yes
Stopping condition	Yes	Yes	Yes	Yes	Yes	Yes

Table 4.6: Parameter-component values for the best performing heuristics at 30D: part 1.

	Sawtooth	Ackley	Sphere	Rosenbrock	General
Group size	10	3	2	9	4
Mutation	Gaussian	None	None	None	Gaussian
Mutation prob	0.3329	na	na	na	0.3197
Network	Global	Global	Hierarch	Global	Ring (n=2)
Baseline	Inertia	Inertia	Inertia	Constrict	Inertia
C1 (base)	0.0063	0.8054	1.1376	2.0501	1.0497
C2 (base)	0.4673	0.3509	1.0335	2.3880	0.5170
ω (base)	0.2706	0.4603	0.1412	na	0.1780
Movement	LEH	DD+LEH	DD	DD+LEH	DD
C3 (DD)	na	4.7125	2.8093	6.6843	8.0436
σ (DD)	na	0.2265	0.1360	0.2451	0.2987
σ_{limit} (DD)	na	0.0066	0.0012	0.0054	0.0100
Min step (DD)	na	0.0815	0.0884	0.0696	0.0568
r3 (DD)	na	1	1	1	1
Relocation (LEH)	LEH	LEH	na	LEH	na
GA pop (LEH)	25	20	na	25	na
GA gens (LEH)	4	5	na	4	na
GA mut prob (LEH)	0.0171	0.2970	na	0.6545	na
GA mut perc (LEH)	0.3692	0.0401	na	0.0397	na
GA elites (LEH)	2	3	na	3	na
GA tour size (LEH)	3	7	na	2	na
Dormancy limit (LEH)	5	4	na	2	na
Inner extent	6	4	15	6	4
Inner max	Random	Random	Random	Random	Random
Use n'hood: dormancy	No	Yes	No	Yes	Yes
Use n'hood: pBest	Yes	Yes	Yes	No	Yes
Stopping condition	Yes	Yes	Yes	Yes	Yes

Table 4.7: Parameter-component values for the best performing heuristics at 30D: part 2.

	Rastrigin	Multi F1	Multi F2	Brankes	Pickel	Heavi
Group size	4	8	8	15	9	8
Mutation	Gaussian	Uniform	Gaussian	Uniform	None	Gaussian
Mutation prob	1.0000	0.1265	1.0000	0.1486	na	0.3022
Network	Global	Global	von Neu	Hierarch	Hierarch	Global
Baseline	Constrict	Inertia	Inertia	Inertia	Inertia	Inertia
C1 (base)	2.2924	2.0730	1.2724	0.4344	0.8688	1.1323
C2 (base)	2.2617	0.6844	1.2904	1.9297	0.9148	0.9203
ω (base)	na	0.5676	0.4673	0.6612	0.8136	0.5147
Movement	DD+ LEH	DD+ LEH	DD+ LEH	LEH	DD+ LEH	DD+ LEH
C3 (DD)	3.9960	6.3242	6.7039	na	7.0708	5.5855
σ (DD)	0.2796	0.2667	0.1695	na	0.1701	0.2226
σ_{limit} (DD)	0.0096	0.0057	0.0063	na	0.0062	0.0069
Min step (DD)	0.0970	0.0575	0.0929	na	0.0953	0.0870
r3 (DD)	1	1	1	na	1	1
Relocation (LEH)	LEH	LEH	LEH	LEH	LEH	LEH
GA pop (LEH)	25	25	25	10	25	25
GA gens (LEH)	4	4	4	10	4	4
GA mut prob (LEH)	0.1569	0.5122	0.5615	0.9996	0.0243	0.0683
GA mut perc (LEH)	0.0049	0.0348	0.0517	0.4723	0.4678	0.0685
GA elites (LEH)	2	1	2	1	3	2
GA tour size (LEH)	4	9	5	3	9	6
Dormancy limit (LEH)	1	5	5	3	5	2
Inner extent	9	4	5	4	4	4
Inner max	Random	Random	Random	PSO	Random	Random
PSO swarm (inner)	na	na	na	2	na	na
PSO C1 (inner)	na	na	na	0.6492	na	na
PSO C2 (inner)	na	na	na	0.1359	na	na
PSO ω (inner)	na	na	na	0.8361	na	na
Use n'hood: dormancy	Yes	No	Yes	No	Yes	Yes
Use n'hood: pBest	Yes	Yes	Yes	No	Yes	No
Stopping condition	Yes	Yes	Yes	Yes	Yes	Yes

Table 4.8: Parameter-component values for the best performing heuristics at 100D: part 1.

	Sawtooth	Ackley	Sphere	Rosenbrock	General
Group size	21	11	5	5	8
Mutation	Gaussian	None	None	Gaussian	None
Mutation prob	0.1605	na	na	0.0141	na
Network	von Neu	Global	Global	Global	Global
Baseline	Inertia	Inertia	Inertia	Inertia	Inertia
C1 (base)	0.9865	2.0713	0.1852	0.1882	0.8688
C2 (base)	1.0166	1.0798	0.9672	0.9148	0.9148
ω (base)	0.3898	0.5652	0.6568	0.5731	0.8136
Movement	DD+ LEH	DD+ LEH	DD+ LEH	DD+ LEH	DD+ LEH
C3 (DD)	8.7102	5.6421	5.0272	8.3429	7.1878
σ (DD)	0.1420	0.3072	0.2198	0.1341	0.1265
σ_{limit} (DD)	0.0049	0.0081	0.0049	0.0039	0.0062
Min step (DD)	0.0669	0.0390	0.0954	0.0964	0.0477
r3 (DD)	1	Random	1	Random	1
Relocation (LEH)	LEH	LEH	LEH	LEH	LEH
GA pop (LEH)	20	20	20	25	25
GA gens (LEH)	5	5	5	4	4
GA mut prob (LEH)	0.0307	0.0871	0.0404	0.3878	0.0110
GA mut perc (LEH)	0.4841	0.0341	0.1861	0.0415	0.1951
GA elites (LEH)	2	1	1	2	3
GA tour size (LEH)	2	2	4	3	10
Dormancy limit (LEH)	2	2	2	1	3
Inner extent	4	4	8	8	4
Inner max	Random	PSO	Random	Random	Random
PSO swarm (inner)	na	2	na	na	na
PSO C1 (inner)	na	2.0158	na	na	na
PSO C2 (inner)	na	1.0902	na	na	na
PSO ω (inner)	na	0.3478	na	na	na
Use n'hood: dormancy	Yes	Yes	Yes	Yes	Yes
Use n'hood: pBest	Yes	Yes	Yes	Yes	Yes
Stopping condition	Yes	Yes	Yes	Yes	Yes

Table 4.9: Parameter-component values for the best performing heuristics at 100D: part 2.

Chapter 5

Conclusion

5.1 Summary

The use of models to support informed decision making is ubiquitous in many real-world situations, as is the desire to identify optimal model solutions. The potential impact of uncertainty on any identified optimal solution is also a common consideration. Here our interest is in the development of improved optimisation methaeuristics for general problems, taking account of implementation uncertainty. Specifically we consider a robust setting, and seek to develop improved methaeuristics for black-box robust optimisation problems.

We assume a classic worst case or min max problem, comprising a local uncertainty neighbourhood inner maximisation search within an outer global minimisation search. We consider a Γ-radius uncertainty set, where the uncertainty neighbourhood around any point is a hypersphere completely defined by a single radius value, Γ. We also assume some limit on the number of function evaluations or model runs available to the decision maker.

Compared to both stochastic optimisation and robust mathematical programming settings, heuristics for black-box robust problems have been less widely considered. In particular general approaches which do not place additional assumptions on the problems, such as limiting the dimension of the problem or making simplifying assumptions about the nature of the objective function surface, have been much less widely considered.

In the first paper we develop a new heuristic for application to the min max problem. This largest empty hypersphere approach takes the local descent directions [BNT10b] concept of moving away from poor, 'high cost points', to a global setting. LEH is an individual-based approach which proceeds by repeatedly identifying regions of the solution space completely devoid of any high cost points and stepping to the centres of these regions. An analysis of alternative approaches for identifying the centre of the largest empty hypersphere identified an optimisation heuristic as the most valid, practical approach. Here a genetic algorithm is employed. A key element of the approach is the use of a 'stopping condition' whereby an inner maximisation is terminated early if it can be determined that a candidate point cannot improve on the current estimate of the global robust optimum.

The new technique was tested on eight established multi-dimensional test problems, across five dimensions between 2D and 100D, and assuming a budget of 10,000 function evaluations. Comparisons were made against a re-starting descent directions and a brute force particle swarm optimisation approach. In all cases random sampling was employed for the inner maximisation layer. Parameter tuning using a genetic algorithm was employed at the heuristic level. At high dimensions the new approach substantially outperforms the comparators for all problems, whilst in other cases it also performs competitively.

In the second paper we develop a new robust heuristic framework using particle swarm optimisation as a basis. This new population based approach extends a baseline robust PSO heuristic through the incorporation of elements of the descent direction and largest empty hypersphere heuristics. The first new capability includes the calculation of particle level d.d. vectors and adds this to the standard inertia based PSO velocity formulation, as a third component. The second new capability includes particle level stopping conditions, an assessment of 'dormant' particles, and relocation by LEH. The framework can be set to employ d.d. and LEH features in combination, or either one individually.

The new framework was tested on ten multi-dimensional test problems, across six dimensions between 2D and 100D. Comparisons were made against a re-starting d.d. approach, a robust PSO heuristic, and LEH. A budget of 5,000 function evaluations was used, with inner maximisation again due to random sampling for all heuristics. Using a genetic algorithm parameters were tuned at the heuristic level, separately for each dimension. The new approach outperforms the comparators at all dimensions, and in particular at 10D, 30D and 60D. The PSO formulation augmented by LEH features alone performs best, followed by the framework employing both LEH and d.d. features together, and then PSO with d.d. only.

The third paper adopts a different approach to the development of improved search algorithms, the automatic generation of heuristics. We employ grammar-guided genetic programming to evolve populations of heuristics for robust problems, employing a tree-based approach and identifying the best algorithms generated in this way. The approach requires the specification and generation of a number of algorithmic building blocks which, when combined appropriately, form complete heuristics. This is our grammar, and includes components offering alternatives for the inner and outer layers, networks for information sharing, and movement formulations. In addition to the development of improved search algorithms, the large numbers of heuristics generated by the GP process enables an analysis of the alternatives employed in each building block against heuristics performance.

The GP process was undertaken using ten established multi-dimensional test problems, at 30D and 100D. A budget of 2,000 function evaluations was assumed. In total twenty two GP runs were undertaken, evolving heuristics targeted separately at each of the ten problems plus a single general ten-case run, for both dimensions. Comparisons were made against the best results from the second paper. The best new general heuristics outperform the comparators, substantially for some of the test problems. Results for the best new individual case heuristics also indicate a strong performance. For the building block level analysis, an inner maximisation by random sampling on a small number of points and using a particle level stopping condition is effective. For the outer minimisation small numbers of particles, a Global topology, and particle movement using a baseline inertia velocity equation augmented by LEH and d.d. capabilities are all effective.

5.2 Further Work

There remains considerable potential for developing improved heuristics to tackle the min max robust problem as set out here.

An obvious way to expand on the work in our third paper is to extend the grammar, and therefore the the sub-algorithmic solution space, on which we apply genetic programming. One direction this could take would be to move to a more general agent based framework, beyond the current PSO baseline. In the first instance considering formulations from other existing population based search heuristics would introduce new movement and information sharing building blocks for the outer minimisation layer. Potential adaptations of these capabilities might then also become apparent, beyond just the use of existing formulations.

A budget on the number of model runs imposes some trade-off on the optimisation search, between each inner maximisation uncertainty neighbourhood analysis and the global exploration of the robust objective solution space. In this context the extent and nature of the inner layer warrants further investigation. One approach would be to build upon the inner maximisation by random sampling in our GP grammar through the use of alternative and more efficient sampling techniques as additional building block components, see for example [Bra01, BF16, FBG19].

Further assumptions used in the formulations here could be expanded upon. For example alternatives to the use of an invisible boundary condition in the baseline PSO algorithm could be added to the grammar, see [RR04, HBM13].

Another consideration is the use of emulation, for either the outer minimisation or inner maximisation layers. Within our literature review in Section 1.3.3 we discuss multiple such approaches. Introducing algorithmic building blocks into our grammar based directly on such emulation approaches, as well as potential adaptations directed towards agent based approaches to the solution of the robust min max problem, could be considered.

At a higher level we could investigate the potential for alternative approaches to the automatic generation of heuristics for robust problems, to the current grammar-guided genetic programming approach described in our third paper.

Finally we might want to expand on some of the limiting assumptions of our work, to consider for example forms of uncertainty beyond those described here. Consideration could be given to uncertainty neighbourhoods other than one defined by the single Γ-radius parameter. Alternatively explicit consideration could be given to model uncertainty, in addition to implementation uncertainty. Expanding the scope of our approaches in these ways could further lead to a direct application to some of the real-world problems mentioned in Section 1.1.

Bibliography

[ABV09] H. Aissi, C. Bazgan, and D. Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427 – 438, 2009.

[AVCMSdCM11] C. Arbex Valle, L. Conegundes Martinez, A. Salles da Cunha, and G. R. Mateus. Heuristic and exact algorithms for a min–max selective vehicle routing problem. *Computers & Operations Research*, 38(7):1054 – 1065, 2011.

[BBC11] D. Bertsimas, D. B. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.

[BF16] J. Branke and X. Fei. Efficient sampling when searching for robust solutions. In J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, editors, *Parallel Problem Solving from Nature – PPSN XIV*, pages 237–246, Cham, 2016. Springer International Publishing.

[BGH$^+$13] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[BHK$^+$09] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. *Exploring Hyper-heuristic Methodologies with Genetic Programming*, pages 177–201. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[BL15] J. Branke and K. Lu. Finding the trade-off between robustness and worst-case quality. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, page 623–630, New York, NY, USA, 2015. Association for Computing Machinery.

[BLS13] I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Inf. Sci.*, 237:82–117, July 2013.

[BN10] D. Bertsimas and O. Nohadani. Robust optimization with simulated annealing. *Journal of Global Optimization*, 48(2):323–334, 2010.

[BNT07] D. Bertsimas, O. Nohadani, and K. M. Teo. Robust optimization in electromagnetic scattering problems. *Journal of Applied Physics*, 101(7):074507, 2007.

[BNT10a] D. Bertsimas, O. Nohadani, and K. M. Teo. Nonconvex robust optimization for problems with constraints. *INFORMS journal on computing*, 22(1):44–58, 2010.

[BNT10b] D. Bertsimas, O. Nohadani, and K. M. Teo. Robust optimization for unconstrained simulation-based problems. *Operations Research*, 58(1):161–178, 2010.

[Bra98] J. Branke. Creating robust solutions by means of evolutionary algorithms. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature — PPSN V*, pages 119–128, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[Bra01] J. Branke. Reducing the sampling variance when searching for robust solutions. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO'01, page 235–242, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[BS04] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[BS07] H-G. Beyer and B. Sendhoff. Robust optimization–a comprehensive survey. *Computer methods in applied mechanics and engineering*, 196(33):3190–3218, 2007.

[BSS01] J. Branke, C. Schmidt, and H. Schmeck. Efficient fitness estimation in noisy environments. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO'01, page 243–250, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[BTBB10] A. Ben-Tal, D. Bertsimas, and D. Brown. A soft robust model for optimization under ambiguity. *Operations research*, 58(4-part-2):1220–1234, 2010.

[BTEGN09] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization.* Princeton University Press, Princeton and Oxford, 2009.

[BTGGN03] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Math. Programming A*, 99:351–376, 2003.

[BTN98] A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805, 1998.

[CBP15] C. Contreras-Bolton and V. Parada. Automatic design of algorithms for optimization problems. In *Computational Intelligence (LA-CCI), 2015 Latin America Congress on*, pages 1–5. IEEE, 2015.

[CG16] A. Chassein and M. Goerigk. A bicriteria approach to robust optimization. *Computers & Operations Research*, 66:181 – 189, 2016.

[Cha93] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10(4):377–409, Dec 1993.

[CK02] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Trans. Evol. Comp*, 6(1):58–73, February 2002.

[CLSS17] R. Chen, B. Lucier, Y. Singer, and V. Syrgkanis. Robust optimization for non-convex objectives. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 4708–4717, USA, 2017. Curran Associates Inc.

[CSZ09] A. M. Cramer, S. D. Sudhoff, and E. L. Zivi. Evolutionary algorithms for minimax problems in robust design. *IEEE Transactions on Evolutionary Computation*, 13(2):444–453, 2009.

[CZ09] X. Chen and Y. Zhang. Uncertain linear programs: Extended affinely adjustable robust counterparts. *Operations Research*, 57(6):1469–1482, 2009.

[dCBF09] D. F. de Carvalho and C. J. A. Bastos-Filho. Clan particle swarm optimization. *International Journal of Intelligent Computing and Cybernetics*, 2(2):197, 2009.

[Din17] A. Dinno. *dunn.test: Dunn's Test of Multiple Comparisons Using Rank Sums*, 2017. R package version 1.3.5.

[Dip10] C. J. Dippel. Using particle swarm optimization for finding robust optima. Technical report, Natural Computing Group, Universiteit Leiden, 2010.

[EDHX17] J. Esteban Diaz, J. Handl, and D-L. Xu. Evolutionary robust optimization in production planning – interactions between number of objectives, sample size and choice of robustness measure. *Computers & Operations Research*, 79:266 – 278, 2017.

[EHG05] E. Elbeltagi, T. Hegazy, and D. Grierson. Comparison among five evolutionary-based optimization algorithms. *Advanced engineering informatics*, 19(1):43–53, 2005.

[EK95] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Oct 1995.

[EKS17] G. Eichfelder, C. Krüger, and A. Schöbel. Decision uncertainty in multiobjective optimization. *Journal of Global Optimization*, 69(2):485–510, 2017.

[Eng12] A. Engelbrecht. Particle swarm optimization: Velocity initialization. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, June 2012.

[ES12] A. E. Eiben and S. K. Smit. *Evolutionary Algorithm Parameters and Methods to Tune Them*, pages 15–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[FBG19] X. Fei, J. Branke, and N. Gülpınar. New sampling strategies when searching for robust solutions. *IEEE Transactions on Evolutionary Computation*, 23(2):273–287, April 2019.

[GH19] M. Goerigk and M. Hughes. Representative scenario construction and preprocessing for robust combinatorial optimization problems. *Optimization Letters*, 13(6):1417–1431, Sep 2019.

[GLT97] B. L. Golden, G. Laporte, and É. D. Taillard. An adaptive memory heuristic for a class of vehicle routing problems with minmax objective. *Computers & Operations Research*, 24(5):445 – 452, 1997.

[GMT14] V. Gabrel, C. Murat, and A. Thiele. Recent advances in robust optimization: An overview. *European journal of operational research*, 235(3):471–483, 2014.

[Goe12] M. Goerigk. *Algorithms and Concepts for Robust Optimization.* PhD thesis, Universität Göttingen, 2012.

[GS10] J. Goh and M. Sim. Distributionally robust optimization and its tractable approximations. *Operations Research*, 58(4-part-1):902–917, 2010.

[GS16] M. Goerigk and A. Schöbel. Algorithm engineering in robust optimization. In L. Kliemann and P. Sanders, editors, *Algorithm Engineering: Selected Results and Surveys*, volume 9220 of LNCS State of the Art of *Lecture Notes in Computer Science*, pages 245–279. Springer Berlin / Heidelberg, 2016.

[HBM13] S. Helwig, J. Branke, and S. Mostaghim. Experimental analysis of bound handling techniques in particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 17(2):259–271, 2013.

[HdMB14] T. Homem-de Mello and G. Bayraksan. Monte carlo sampling-based methods for stochastic optimization. *Surveys in Operations Research and Management Science*, 19(1):56 – 85, 2014.

[Her99] J. W. Herrmann. A genetic algorithm for minimax optimization problems. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 2, pages 1099–1103. IEEE, 1999.

[HGD20a] M. Hughes, M. Goerigk, and T. Dokka. Automatic generation of robust particle swarm optimisation metaheuristics using grammar-guided genetic programming. *arXiv e-prints*, page arXiv:2004.07294, Apr 2020.

[HGD20b] M. Hughes, M. Goerigk, and T. Dokka. Particle swarm metaheuristics for robust optimisation with implementation uncertainty. *arXiv e-prints*, page arXiv:2003.09664, Mar 2020.

[HGW19] M. Hughes, M. Goerigk, and M. Wright. A largest empty hypersphere metaheuristic for robust optimisation with implementation uncertainty. *Computers & Operations Research*, 103:64 – 80, 2019.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition.* Springer Series in Statistics. Springer New York, 2009.

[Jen04] M. T. Jensen. *A New Look at Solving Minimax Problems with Coevolutionary Genetic Algorithms*, pages 369–384. Springer US, Boston, MA, 2004.

[JJB07] D. Jakobović, L. Jelenković, and L. Budin. Genetic programming heuristics for multiple machine scheduling. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar, editors, *Genetic Programming*, pages 321–330, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[JM05] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(6):1272–1282, Dec 2005.

[JSW98] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, December 1998.

[JY13] M. Jamil and X-S. Yang. A literature survey of benchmark functions for global optimization problems. *International Journal of Mathematical Modelling and Numerical Optimisation (IJMMNO)*, 4(2):150–194, 2013.

[Kam09] K. Kameyama. Particle swarm optimization - a survey. *IEICE Transactions on Information and Systems*, E92.D(7):1354–1361, 2009.

[KE95] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia, 1995.

[KEB10] J. Kruisselbrink, M. Emmerich, and T. Bäck. An archive maintenance scheme for finding robust solutions. In R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, pages 214–223, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[KES01] J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm intelligence.* Morgan Kaufmann, 2001.

[Kir17] M. S. Kiran. Particle swarm optimization with a new update mechanism. *Applied Soft Computing*, 60:670 – 678, 2017.

[KM02] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1671–1676 vol.2, May 2002.

[Koz92] J.R. Koza. *Genetic Programming: On the Programming of Computers* *by Means of Natural Selection*. A Bradford book. MIT Press, 1992.

[KRD$^+$11] J. W. Kruisselbrink, E. Reehuis, A. Deutz, T. Bäck, and M. Emmerich. Using the uncertainty handling cma-es for finding robust optima. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 877–884, New York, NY, USA, 2011. ACM.

[Kru12] J. W. Kruisselbrink. *Evolution strategies for robust optimization*. PhD thesis, Leiden Institute of Advanced Computer Science (LIACS), Faculty of Science, Leiden university, 2012.

[KVDHL16] K. Khac Vu, C. D'Ambrosio, Y. Hamadi, and L. Liberti. Surrogate-based methods for black-box optimization. *International Transactions in Operational Research*, 24(3):393–424, 2016.

[KY97] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Kluwer Academic Publishers, 1997.

[LBR16] K. Lu, J. Branke, and T. Ray. Improving efficiency of bi-level worst case optimization. In J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, editors, *Parallel Problem Solving from Nature – PPSN XIV*, pages 410–420, Cham, 2016. Springer International Publishing.

[Luk13] S. Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[MHW$^+$10] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3):365–396, Sep 2010.

[Mit98] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[MKA11] K. Masuda, K. Kurihara, and E. Aiyoshi. A novel method for solving min-max problems by using a modified particle swarm optimization. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 2113–2120. IEEE, 2011.

[MKN03] R. Mendes, J. Kennedy, and J. Neves. Watch thy neighbor or how the swarm can learn from its environment. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*, pages 88–94, April 2003.

[MLIDLS14] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & Operations Research*, 51:190 – 199, 2014.

[MLM15] S. Mirjalili, A. Lewis, and S. Mostaghim. Confidence measure: A novel metric for robust meta-heuristic optimisation algorithms. *Information*

Sciences, 317:114 – 142, 2015.

[MP16] P. B. C. Miranda and R. B. C. Prudêncio. Tree-based grammar genetic programming to evolve particle swarm algorithms. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 25–30, Oct 2016.

[MP17] P. B.C. Miranda and R. B.C. Prudêncio. Generation of particle swarm optimization algorithms: An experimental study using grammar-guided genetic programming. *Applied Soft Computing*, 60:281 – 296, 2017.

[MPF09] Bader. M., R. Poli, and S. Fatima. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, 1(3):205–219, 2009. Funders: EPSRC.

[MWPL13] J. Marzat, E. Walter, and H. Piet-Lahanier. Worst-case global optimization of black-box functions through kriging and relaxation. *Journal of Global Optimization*, 55(4):707–727, Apr 2013.

[MWPL16] J. Marzat, E. Walter, and H. Piet-Lahanier. A new expected-improvement algorithm for continuous minimax optimization. *Journal of Global Optimization*, 64(4):785–802, 2016.

[MZ18] Y. Mei and M. Zhang. Genetic programming hyper-heuristic for multi-vehicle uncertain capacitated arc routing problem. pages 141–142, 07 2018.

[Nah17] C. Nahr. Tektosyne library for java, 2017. Available at http://www.kynosarges.org.

[NMES11] A. Nickabadi, M. Mehdi Ebadzadeh, and R. Safabakhsh. A novel particle swarm optimization algorithm with adaptive inertia weight. *Applied Soft Computing*, 11(4):3658 – 3670, 2011.

[Noh11] A. Nohejl. Grammar-based genetic programming. Master's thesis, Department of Software and Computer Science Education, Faculty of Mathematics and Physics, Charles University in Prague, 2011.

[ONL06] Y-S. Ong, P. B. Nair, and K. Y. Lum. Max-min surrogate-assisted evolutionary algorithm for robust design. *IEEE Transactions on Evolutionary Computation*, 10(4):392–404, 2006.

[OS97] A. Okabe and A. Suzuki. Locational optimization problems solved through voronoi diagrams. *European Journal of Operational Research*, 98(3):445 – 456, 1997.

[PBJ06] I. Paenke, J. Branke, and Y. Jin. Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation. *IEEE Transactions on Evolutionary Computation*, 10(4):405–420, 2006.

[PDCL05] R. Poli, C. Di Chio, and W. B. Langdon. Exploring extended particle swarms: A genetic programming approach. In *Proceedings of the*

7th Annual Conference on Genetic and Evolutionary Computation, GECCO '05, pages 169–176, New York, NY, USA, 2005. ACM.

[PF10] G. L. Pappa and A. A. Freitas. *Automating the design of data mining algorithms: an evolutionary computation approach.* Natural Computing Series. Springer, January 2010.

[PHP12] P. Pošík, W. Huyer, and L. Pál. A comparison of global search algorithms for continuous black box optimization. *Evolutionary computation*, 20(4):509–541, 2012.

[POH$^+$14] G. L. Pappa, G. Ochoa, M. R. Hyde, A. A. Freitas, J. Woodward, and J. Swan. Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 15(1):3–35, 2014.

[R C19] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2019.

[RC13] S. Roy and S. Chaudhuri. Bio-inspired ant algorithms: A review. *International Journal of Modern Education and Computer Science*, 5(4):25, 2013.

[RR04] J. Robinson and Y. Rahmat-Samii. Particle swarm optimization in electromagnetics. *IEEE Transactions on Antennas and Propagation*, 52(2):397–407, Feb 2004.

[SAMO03] M. Stephenson, S. Amarasinghe, M. Martin, and U-M. O'Reilly. Meta optimization: Improving compiler heuristics with machine learning. *SIGPLAN Not.*, 38(5):77–90, May 2003.

[SBP18] S. Sengupta, S. Basak, and R. Peters. Particle swarm optimization: A survey of historical and recent developments with hybridization perspectives. *Machine Learning and Knowledge Extraction*, 1(1):157–191, 2018.

[SE98] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 69–73, 1998.

[SEFR19] N. D. Sanders, R. M. Everson, J. E. Fieldsend, and A. A. M. Rahat. A Bayesian Approach for the Robust Optimisation of Expensive-To-Evaluate Functions. *arXiv e-prints*, page arXiv:1904.11416, Apr 2019.

[SK02] Y. Shi and R. Krohling. Co-evolutionary particle swarm optimization to solve min-max problems. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1682–1687. IEEE, 2002.

[SP97] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of*

Global Optimization, 11(4):341–359, 1997.

[Tal09] E-G. Talbi. *Metaheuristics: From Design to Implementation.* Wiley Publishing, 2009.

[TG97] S. Tsutsui and A. Ghosh. Genetic algorithms with a robust solution searching scheme. *IEEE transactions on Evolutionary Computation*, 1(3):201–208, 1997.

[Tou83] G. T. Toussaint. Computing largest empty circles with location constraints. *International Journal of Computer & Information Sciences*, 12(5):347–358, 1983.

[uRL17] S. ur Rehman and M. Langelaar. Expected improvement based infill sampling for global robust optimization of constrained problems. *Optimization and Engineering*, 18(3):723–753, Sep 2017.

[uRLvK14] S. ur Rehman, M. Langelaar, and F. van Keulen. Efficient kriging-based robust optimization of unconstrained problems. *Journal of Computational Science*, 5(6):872 – 881, 2014.

[vLHVB$^+$12] R. R.S. van Lon, T. Holvoet, G. Vanden Berghe, T. Wenseleers, and J. Branke. Evolutionary synthesis of multi-agent systems for dynamic dial-a-ride problems. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, page 331–336, New York, NY, USA, 2012. Association for Computing Machinery.

[Whi95] P. A. Whigham. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, USA, 1995. University of Rochester.

[WYO16] L. Wang, B. Yang, and J. Orchard. Particle swarm optimization using dynamic tournament topology. *Appl. Soft Comput.*, 48(C):584–596, November 2016.

[ZWJ15] Y. Zhang, S. Wang, and G. Ji. A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical Problems in Engineering*, 2015.

[ZZ10] A. Zhou and Q. Zhang. A surrogate-assisted evolutionary algorithm for minimax optimization. In *IEEE Congress on Evolutionary Computation*, pages 1–7, July 2010.