Automatic generation of algorithms for robust optimisation problems using Grammar-Guided Genetic Programming

Hughes, M., Goerigk, M., & Dokka, T. (2021). Automatic generation of algorithms for robust optimisation problems using Grammar-Guided Genetic Programming. *Computers & Operations Research, 133*, [105364]. https://doi.org/10.1016/j.cor.2021.105364

**Published in:**
Computers & Operations Research

**Document Version:**
Publisher's PDF, also known as Version of record

**Queen's University Belfast - Research Portal:**
Link to publication record in Queen's University Belfast Research Portal

**Publisher rights**
Copyright 2021 the authors.
This is an open access article published under a Creative Commons Attribution License (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution and reproduction in any medium, provided the author and source are cited.

**General rights**
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

**Take down policy**
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person’s rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.
Automatic generation of algorithms for robust optimisation problems using Grammar-Guided Genetic Programming

Martin Hughes a,*, Marc Goerigk b, Trivikram Dokka a,c

a Department of Management Science, Lancaster University, United Kingdom
b Network and Data Science Management, University of Siegen, Germany
c Management School, Queens University Belfast, United Kingdom

ARTICLE INFO

Keywords:
Robust optimisation
Implementation uncertainty
Metaheuristics
Global optimisation
Genetic programming

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.

1. Introduction

Models are routinely used to support informed decision making in many real-world settings. A model enables the user to simulate the problem at hand and consider the impact of changing decision variables, when it is impractical to apply such analysis directly to the real problem. Furthermore, in order to identify good decision variable values, optimisation search techniques may be employed to efficiently investigate the decision variable solution space. However such a search may be impacted by issues like model run times, the size of the solution space, and uncertainty, see Ben-Tal et al. (2009) and Goerigk and Schöbel (2016). 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 programme, 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, for example see Paenke et al. (2006), Bertsimas et al. (2007, 2010b), Kruisselbrink (2012), Gabrel et al. (2014) and Goerigk and Schöbel (2016).

When it is known how the uncertainty is distributed, the problem may be considered one of stochastic optimisation, see Paenke et al. (2006) and de Mello and Bayraksan (2014). In contrast 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 (Ben-Tal and Nemirovski, 1998). We seek improved methods for robust general, black-box problems that
can be employed in this setting. This requirement is beyond the scope of robust mathematical programming methods, necessitating the use of a metaheuristic approach. The simplistic alternative of ignoring the uncertainty and using a classic optimisation method can produce suboptimal results, see Ben-Tal et al. (2009) and Goerigk and Schöbel (2016).

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. Burke et al. (2013) and Pappa et al. (2014). This is the approach adopted here. Such an approach has been applied to a number of practical problems, for example computing Stephenson et al. (2003), machine scheduling (Jakobović et al., 2007), timetabling (Bader et al., 2009), the design of data mining algorithms (Pappa and Freitas, 2010), and agent-based transportation simulation (van Lon et al., 2012).

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 domain is known and there is an appreciation of model run time limitations, the latter leading to some budget on model runs with the associated assumption that model run time is the limiting factor here and any search heuristic processing times are negligible by comparison. These are reasonable assumptions for some real-world problems, but inevitably the restriction on model runs will limit both the global explorative ability of the search and any localised assessment of uncertainty neighbourhoods. 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. 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) (Nohejil, 2011). 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 operations. 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) (Nohejil, 2011).

As with any evolutionary approach, GP employs combination and mutation operations to generate improved (fitter) solutions. However integrating sub-algorithms (computer sub-programmes 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 (Mascia et al., 2014; Contreras-Bolton and Parada, 2015; Miranda and Prudêncio, 2016; Miranda and Prudêncio, 2017). 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 (Miranda and Prudêncio, 2016).

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 2000 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 topology. The preferred particle movement uses an inertia based velocity equation plus specialised capabilities drawn from the largest empty hypersphere (Hughes et al., 2019, 2020) and descent directions (Bertsimas et al., 2007, 2010a,b; Hughes et al., 2020) heuristics.

In Section 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 3 gives an overview on the automatic generation of algorithms, and in Section 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 5 describes the experimental analysis, results for the best heuristics identified, and an analysis of heuristic sub-component performance. Section 6 provides conclusions and possible directions for future work.

2. Robust optimisation

2.1. Problem description

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

\[
\min f(x) \quad \text{s.t. } x \in \mathcal{X}
\]

This is the nominal problem. The objective \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) operates on the \( n \)-dimensional vector of decision variables \( x = (x_1, x_2, \ldots, x_n)^T \) in the feasible region \( \mathcal{X} \subseteq \mathbb{R}^n \). Here we assume box constraints \( \mathcal{X} = \prod_{i=1}^n [u_i, l_i] \). Any other feasibility constraint is assumed to be dealt with by a penalty in the objective. The notation \( [a] := \{1, \ldots, n\} \) is used. Consider the problem due to Kruisselbrink (2012) in Fig. 1, where \( \mathcal{X} \subseteq \mathbb{R}^3 \), \( l_1 = 0 \) and \( u_1 = 10 \). The objective function of the nominal problem is the black curve.

Introducing uncertainty \( \Delta x \) around the intended solution \( x \), makes only a solution \( \hat{x} = x + \Delta x \) achievable. If it is assumed that the uncertainty neighbourhood around a candidate is completely defined by a radius \( \Gamma > 0 \), aligning with the uncertainty description employed in Bertsimas et al. (2007, 2010a,b) and Hughes et al. (2019, 2020), the uncertainty set is:

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

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

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

In Fig. 1, \( \Gamma = 0.5 \) and each point on the worst case cost (dashed grey) curve \( g(x) \) is generated by finding the maximum value on the nominal curve within a range of \(-0.5\) to \(+0.5\) of the desired solution \( x \).
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 $x \in \mathcal{X}$ for the nominal objective function:

$$\min_{x \in \mathcal{X}} g(x) = \min_{x \in \mathcal{X}} \max_{\Delta x \in \mathcal{U}} f(x + \Delta x) \quad (4)$$

In the example this moves the global minimisation search from the black curve to the grey curve, where the global optimum shifts from $x_0$ to $x'_0$.

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

The consideration of a $\Gamma$-radius uncertainty neighbourhood does not preclude extending the search techniques developed here to encompass other descriptions of uncertainty.

### 2.2. Existing solution methods

Since its initial formalisation (Kouvelis and Yu, 1997; Ben-Tal and Nemirovski, 1998) robust optimisation has been heavily aligned with mathematical programming, see Bertsimas et al. (2011), Gabrel et al. (2014) and Goerigk and Schöbel (2016). 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 Marzat et al. (2013), Goerigk and Schöbel (2016) and Marzat et al. (2016).

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. Hughes et al. (2019) and Hughes et al. (2020), more refined robust-specific methods may be desirable. Such techniques include co-evolutionary approaches (Herrmann, 1999; Shi and Krohling, 2002; Jensen, 2004; Cramer et al., 2009; Masuda et al., 2011), robust evolutionary approaches Tsutsui and Ghoosh (1997), Beyer and Sendhoff (2007), and the use of emulation by surrogates or meta-models Ong et al. (2006), Beyer and Sendhoff (2007), Zhou and Zhang (2010), Yu et al. (2016) including Kriging (Marzat et al., 2013; ur Rehman et al., 2014; Marzat et al., 2016; ur Rehman and Langelaar, 2017) and Bayesian techniques (Chen et al., 2017; Sanders et al., 2019). 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.

In contrast our interest is in robust metaheuristics for general black-box problems under implementation uncertainty, which can be readily applied to problems of moderate dimension and where no prior knowledge of the nature of the objective function surface is assumed, but where the search is restricted by practical limitations on the number of function evaluations that can be performed. In this context 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 (Bertsimas et al., 2007, 2010a,b) and the global largest empty hypersphere (LEH) method (Hughes et al., 2019). Both are single-solution techniques, although elements of these approaches have been incorporated into robust population-based approaches (Hughes et al., 2020). First, however, we consider the non-robust particle swarm optimisation (PSO) metaheuristic (Kennedy and Eberhart, 1995; Kennedy et al., 2001; Talbi, 2009), 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.

#### 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 many PSO formulations building on this general concept, see for example (Kameyama, 2009; Nickabadi et al., 2011; Zhang et al., 2015; Kiran, 2017; Sengupta et al., 2018).

In a robust setting two-swarm co-evolutionary PSO techniques have been considered (Shi and Krohling, 2002; Masuda et al., 2011), whilst (Hughes et al., 2019) employ a simple robust PSO (rPSO) as a comparator heuristic, with inner maximisation by random sampling. The rPSO from Hughes et al. (2019) is extended in Hughes et al. (2020) through the addition of d.d. and LEH elements. Here the framework for each heuristic in the GP population is a basic PSO formulation (Kennedy and Eberhart, 1995; Kennedy et al., 2001; Talbi, 2009), 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 $t = 0$ at randomly located points $x'(0)$ 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, $x'$. 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 neighbour particle in their entire histories, $x'$. A number of neighbourhood topologies are included as components in the grammar here, see Section 4.2.4.

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

$$x'(t) = x'(t - 1) + v'(t) \quad (5)$$

There are a number of alternative velocity formulations. In the grammar in Section 4.2 two of the most basic formulations are considered, including so-called inertia (Shi and Eberhart, 1998; Kennedy et al., 2001) and constriction (Clerc and Kennedy, 2002; Kennedy et al., 2001) coefficients:

$$v'(t) = \omega \cdot v'(t - 1) + C_1 \cdot r_1 \cdot (x'_j(t) - x'(t - 1)) + C_2 \cdot r_2 \cdot (\hat{x}_j - x'(t - 1)) \quad (6)$$

$$v'(t) = \chi \cdot \left( v'(t - 1) + C_1 \cdot r_1 \cdot (x'_j(t) - x'(t - 1)) + C_2 \cdot r_2 \cdot (\hat{x}_j - x'(t - 1)) \right) \quad (7)$$

where

$$\chi = \frac{2}{2 - \phi - \sqrt{\phi^2 - 4\phi}} \quad (8)$$

In Fig. 1. The worst case cost curve (dashed grey) is generated by determining the maximum objective value in the uncertainty neighbourhood around all points $x$ on the nominal (solid black) curve. Due to the uncertainty the global optimum shifts to $x'_0$.

![Diagram](https://via.placeholder.com/150)
Computers and Operations Research 133 (2021) 105364

with

\[ \phi = C_1 + C_2 \]  

(9)

Here particle velocities \( \mathbf{v}(0) \) are initialised by uniform random sampling \( \sim U(0, 1)^n \) (Engelbrecht, 2012). Each component of the random vectors \( r \) is typically randomly sampled individually, \( r_1, r_2 \sim U(0, 1) \). Vector multiplication is component-wise. The scalar terms \( v \) represen weightings that a particle puts on its \( x_i^t(C_i) \) versus \( x_f(C_f) \) location data. The inertia scalar \( \omega \) tends the significance of the preceding velocity, whilst the constriction scalar \( \chi \) is used to avoid particles ‘exploding’ — disappearing out of the feasible region. Here an invisible boundary condition is adopted (Robinson and Rahmat-Samii, 2004), 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 4.2.6.

The inertia and constriction formulations, Eqs. (6) and (7) respectively, represent an rPSO baseline movement capability. For any given heuristic in our GGGP the movement can be extended through the addition of components, described in Section 4.2. This includes building blocks based on a series of metaheuristics for robust problems developed using rPSO as a framework in Hughes et al. (2020), and novel features here. These developments are largely based around two robust search techniques, d.d. and LEH.

2.2.2. Descent direction

Descent directions is an exploitation-focused, individual-based robust local search technique (Bertsimas et al., 2007, 2010b,a) for solving Eq. (4), although it can easily be extended to approximate a global search through random re-starts each time a local search completes (Hughes et al., 2019). We briefly summarise the method outlined in Bertsimas et al. (2007, 2010b,a).

In d.d. at each candidate point \( x \) in the decision variable space that the search moves to, an inner maximisation search is performed to assess that point’s uncertainty neighbourhood \( N(x) = \{ x + \Delta x | \Delta x \in U \} \) and approximate the worst case cost \( g(x) \approx \tilde{g}(x) \). 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 \( \Gamma \)-radius uncertainty region. At a candidate point \( x \) the high cost set \( H_\sigma(x) \) is defined as:

\[ H_\sigma(x) : = \{ x' \in H \cap N(x) | \tilde{g}(x') \geq \tilde{g}(x) - \sigma \} \]  

(10)

\( \sigma \) is the threshold value for determining what constitutes an hcp.

The optimum (descent) direction originating at the current candidate \( x(t) \) at iteration \( t \), and pointing away from the hcps, is then calculated using mathematical programming. For a direction vector \( d \), the angle \( \theta \) between the vectors connecting the points in \( H_\sigma(x(t)) \) to \( x(t) \), and \( d \), is maximised:

\[ \begin{align*}
\min_{\beta} \quad & \beta \\
\text{s.t.} \quad & |d| \leq 1 \\
& d^T \left( \frac{h - x(t)}{|h - x(t)|} \right) \leq \beta \\
& \forall h \in H_\sigma(x(t)) \\
& \beta \leq -\varepsilon
\end{align*} \]  

(12)

(13)

(14)

Here \( \varepsilon \) is a small positive scalar, so from Eq. (14) \( \beta \) is negative. The left hand side of constraint Eq. (13) is the multiplication of \( \cos \theta \) and \( |d| \), for all hcps in \( H_\sigma(x(t)) \) and a feasible direction \( d \). Eq. (13) therefore relates \( \beta \) to the maximum value for \( \cos \theta \) across all hcps. As the objective Eq. (11) is to minimise \( \beta \), and \( \beta \) is negative, the angle \( \theta \) will be greater than 90° and maximised. Also minimising \( \beta \) in combination with Eq. (12) normalises \( d \). A standard solver such as CPLEX can be used to solve Eqs. (11) to (14). When a feasible direction cannot be found, that is Eqs. (11) to (14) 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 \( x(t) \), in this optimum direction \( d \) with a step size \( \rho(t) \) large enough that the points in \( H_\sigma(x(t)) \) are at a minimum on the boundary of the uncertainty neighbourhood of the next candidate at step \( t + 1 \). Then \( x(t + 1) = x(t) + \rho(t) \cdot d \), where:

\[ \rho(t) = \min \left\{ d^T(h - x(t)) + \sqrt{d^T(h - x(t))^2 + |h - x(t)|^2 + \Gamma^2} | h \in H_\sigma(x(t)) \right\} \]  

(15)

Steps are repeated until a local minimum is reached.

In one of the rPSO variants described in Hughes et al. (2020), given neighbourhood uncertainty information for each particle at each step, a descent direction vector is calculated. This vector \( d(t-1) \) for particle \( j \) at step \( t \) is used for the calculation of an additional velocity component:

\[ C_3, r_3 \cdot d(t-1) \]  

(16)

In Hughes et al. (2020) each component of \( r_1 \) is randomly sampled individually, \( r_1 \sim U(0, 1)^n \), 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, Eq. (6), is then used in Hughes et al. (2020):

\[ \mathbf{v}^j(t) = \omega \cdot \mathbf{v}^j(t-1) + C_1 \cdot r_1 \cdot (x^j(t) - x^j(t-1)) + C_2 \cdot r_2 \cdot (\tilde{x} - x^j(t-1)) + C_3 \cdot r_3 \cdot d(t-1) \]  

(17)

Building blocks components based on this d.d. approach, and associated parameters, are considered in the grammar here. Details are given in Section 4.2.

2.2.3. Largest empty hypersphere

Largest empty hypersphere (LEH) is an exploration-focused individual-based robust global search technique (Hughes et al., 2019) for solving Eq. (4). LEH takes the d.d. concept of hcps to a global setting, identifying a high cost set \( H \), of poor points from within the global history set \( H \), and moving to the centre of the region completely devoid of all such points. \( H \) contains those points in \( H \) with nominal objective function value \( f(x) \) greater than a threshold \( r \). In LEH \( r \) equals the current estimated robust global minimum value.

The centre of the LEH, \( x(t) \in H \) at iteration \( t \), is the estimated point furthest from all hcps in \( H \), and is approximated using a genetic algorithm (GA). Movement from centre of LEH to centre of LEH repeats until no point \( x(t) \in H \) which is at least \( r \) 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 \( x(t) \), however in LEH each objective function evaluation in a neighbourhood analysis \( f(x(t)) + \Delta(x(t)) \) is compared to \( r \), with the inner search terminating if that value exceeds \( r \). This recognises that the current point \( x(t) \) will not 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 Hughes et al. (2020) 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 $x_i$. 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 $x_i$.

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 $x_i$, 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 (Robinson and Rahmat-Samii, 2004). In either case dormancy suggests that a particle has become ‘stuck’. In Hughes et al. (2020) dormant particles are relocated to the centre of the largest empty hypersphere devoid of all hcp, 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 4.2.

3. 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 (Burke et al., 2009, 2013). 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 programmes. 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) (Koza, 1992; Burke et al., 2009). This employs the well known high level concepts of selection, combination and mutation to evolve a population of computer programmes, or in our case search heuristics for robust problems.

When considering the applications of genetic programming to automatically generate optimisation search approaches (Burke et al., 2013), the use of tree-based context-free grammar-guided GP (McKay et al., 2010) to the generation of PSO heuristics described in Miranda and Prudêncio (2016), Miranda and Prudêncio (2017) 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 Mei and Zhang (2018), where GP is applied to a vehicle routing problem including uncertainty. In terms of optimisation for robust problems, Goerigk and Hughes (2019) use a simple GP-based approach to evolve techniques for robust combinatorial optimisation problems. However, to the author's knowledge Goerigk and Hughes (2019) 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. Genetic programming of metaheuristics for robust problems

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 (Koza, 1992; McKay et al., 2010; Nohejl, 2011). 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 (Koza, 1992), 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 4.3. Details of the individual sub-components are given now in Section 4.2.

4.2. Grammar

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 1. 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. What distinguishes the robust PSO form of Algorithm 1 from a non-robust formulation is the inner maximisation, lines 11 to 17. This generates an estimate of the robust objective for each particle at each point visited.

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 1, 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 Fig. 2. The specific approach adopted here is known as context-free grammar genetic
Algorithm 1 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 \( (j \text{ in } 1, \ldots, \text{size}) \) do
4:         if \( (t = 0) \) then
5:            Randomly initialise particle \( x'(0) \in X \)
6:         else
7:            Update particle velocity according to \((\text{velocity}, \text{vel} \text{Params})\)
8:         end if
9:      end for
10:     Update particle position according to \((\text{movement}, \text{move} \text{Params}, \text{topol} \text{Params})\)
11:    end if
12:   if \( (x'(t) \in X) \) then
13:      Perform inner maximization - robust component of PSO search:
14:         for all \( (k \text{ in } 1, \ldots, \text{inExt}) \) do
15:            Select uncertainty neighbourhood point: \((\text{inner}, \text{inner} \text{Params})\)
16:         end for
17:     end if
18:    if \( (\text{budget} = 0) \) then break: to go to end end if
19:   end while
20:  \( t \leftarrow t + 1 \)
21: return Current estimate of robust global best

programming (CFG-GP). This uses a tree-based representation of algorithms and standard tree-based operators in the evolutionary process, see McKay et al. (2010) and Miranda and Prudêncio (2016).

The grammar includes non-terminal nodes, indicated by \(< \text{ >}, \text{ 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 Fig. 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. 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 terminal node that portion of the heuristic is complete. The final heuristic is achieved when there are no more terminal nodes to process in the sequence.

Sub-component details are given in Sections 4.2.2 to 4.2.9. The high level <Outer> and <Inner> elements are identifiable in Fig. 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 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. (Bertsimas et al., 2010b) and LEH (Hughes et al., 2019) techniques and their variants (Hughes et al., 2020). <Inner> is by random sampling, or a PSO or GA search, along with additional <nDorm>, <nPBest> and <Stopping> capabilities, based on features in Hughes et al. (2020) and explained here in Sections 4.2.7 to 4.2.9. There are also a number of sub-components and parameters associated with many of these elements, which in total constitute our grammar.

One way to visualise this process is in the form of a tree (Koza, 1992; Whigham, 1995), Fig. 3, showing the high level structure of a heuristic generated by the production rules in Fig. 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.

<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.

4.2.2. 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 2.2.1, and some system of particle information sharing. The latter, <Network>, is discussed in Section 4.2.4. In all heuristics the <Group> size, <C1> and <C2> parameter values are sampled from the ranges defined in the grammar in Fig. 2. The need for the <w> term depends on the choice of <Baseline>.

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 Miranda and Prudêncio (2017). 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 \( \text{<Prob Mutate>} \). If mutation is confirmed, any given dimension is mutated with probability randomly sampled from between 0 and \( \frac{1}{n} \), 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.

### 4.2.4. 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 \( \text{<Network>} \) component. Of the large number of network topologies available (Kennedy and Eberhart, 1995; Kennedy and Mendes, 2002; Mendes et al., 2003; Janson and Middendorf, 2005; de Carvalho and Bastos-Filho, 2009; Wang et al., 2016; Miranda and Prudêncio, 2017), 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, \( \hat{x}_{\text{best}} \), is shared in a manner defined by the baseline velocity Eqs. (6) or (7).

- **Global**: This is the most basic formulation, with all particles accessing the same neighbourhood information — the current robust global
minimum location (Kennedy and Eberhart, 1995; Kennedy et al., 2001).
• Focal: A single particle is randomly selected as the focal. All particles
access the same neighbourhood information, the focal particle’s
best information, (Kennedy et al., 2001; Kennedy and Mendes, 2002).
• 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 (Kennedy et al., 2001; Kennedy and Mendes, 2002).
• 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 (Kennedy and Mendes,
2002; Miranda and Prudêncio, 2017).
• 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 (de Carvalho and Bastos-Filho,
2009; Miranda and Prudêncio, 2017).
• 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 Mendes
et al. (2003), Miranda and Prudêncio (2017).
• Hierarchical: All particles are randomly arranged in a tree forma-
tion, 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 (Janson and Middendorf, 2005; Miranda
and Prudêncio, 2017).

4.2.5. Building blocks: Additional movement capability
The baseline PSO capability can be augmented by additional move-
ment components based on the descent directions (Bertsimas et al.,
2010b) and largest empty hypersphere (Hughes et al., 2019) ap-
proaches, as proposed in Hughes et al. (2020). The additional formu-
lations 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 2.2.2 and 2.2.3, and Hughes et al.
(2020).

Both <DD> and <LEH> require the determination of further non-
terminal nodes. <DD> employs the nodes: <C3>, <r>, <r> <limit>,
<Min step> and <r3>. These are d.d. parameters whose descriptions
can be found in Bertsimas et al. (2010b), Hughes et al. (2019, 2020).
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. Eqs. (16) and (17) in Section 2.2.2. In the original formulation
each element of $r_3$ is randomly sampled individually, $r_3 \sim U(0, 1)$. This
alternative is available in the component <Random r3> in the
grammar, along with another where each element of $r_3$ 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 re-
location> 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 Hughes et al. (2020), as described in
Section 2.2.3. Here this is designated by <LEH relocation>, and if
selected the parameters <lpop>, <lmutP>, <lmut>, <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 Hughes et al. (2019, 2020). 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.

4.2.6. 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 (Mirjalili et al.,
2017). 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 the non-terminal nodes <In Ext>,
<Form Inner> and <Stopping>. <Form Inner> further generates
the non-terminal nodes <In Type>, <nPBest> and <nPBest>.

<In Ext>, <nPBest> and <Stopping> are discussed in Sec-
Sections 4.2.7 to 4.2.9. <In Ext> and <In Type> relate to the extent and
type 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 <In Type>:
random sampling { }, or inner PSO <In PSO> or genetic
algorithm <In GA> searches. All apply to a candidate point’s F-radius
uncertainty neighbourhood, Section 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 ao>, the parameters for an inertia form of PSO: an
inner swarm size and settings for $C_1$, $C_2$ and $ao$. For an inner PSO the
inertia formulation Eq. (6) is fixed. <In GA> requires the determina-
tion of <In pop>, <In mutP>, <In mut>, <In elites> and <In
tour>, parameters for a standard form of GA (Talbi, 2009): 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.

4.2.7. Building blocks: Dormancy — use of neighbourhood information

The consideration of particle dormancy leading to its relocation (Hughes et al., 2020), is described in Section 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 \(<nDorm>\).

4.2.8. Building blocks: Supplement \(x^*_j\) — 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 \(x^*_j\) as employed in the inertia or constriction velocity formulations, Eqs. (6) or (7). 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 \(<nPBest>\).

4.2.9. 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 Hughes et al. (2020) and Section 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 \(<Stopping>\).

4.3. 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 Fig. 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 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. Eiben and Smit (2012). 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 Fig. 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. Talbi (2009). 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 programmes and algorithms (Poli et al., 2005; McKay et al., 2010; Nohejl, 2011; Miranda and Prudêncio, 2017), 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, Figs. 2 and 3, with standard random tree-based combination and mutation operators (McKay et al., 2010; Miranda and Prudêncio, 2016).

Consider the high-level heuristic tree representation in Fig. 3 with the addition of ‘cut’ points, Fig. 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 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
individual case GP runs and analysis further enable the assessment of the effectiveness of heuristic sub-components, and their commonality to run simulation problem. Furthermore both the general case and the intention that a full GGGP analysis would be applied to an expensive problems to be tackled using an improved heuristic. It is not the 10 individual case GP run all heuristics are applied to a single test problem. For the one general case GP run, all heuristics are applied to all 10 test problems. Furthermore each of the 11 GP runs (10 individual cases plus one general case) are undertaken for both 30D and 100D, giving 22 separate GP runs in total.

Each GP run employs a population of 500 heuristics, evolving them over six generations. At the start of each new generation the four best performing (elite) heuristics are retained intact. The selection process uses a tournament approach with a tournament size of 10 heuristics.

In the GP runs, when a heuristic is applied to a problem the search it undertakes is limited by a budget of 2000 function evaluations, constraining both the global outer minimisation search and the localised inner maximisation searches. In the GP run each individual heuristic is applied to the problem/problems at hand 20 times in order to generate a mean which is used to determine the fitness of the heuristic, within the evolutionary GP process. 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 \( I \)-uncertainty neighbourhood of the optimum.

The intention of considering a general case is to establish an improved general heuristic which could be readily applied to any problem without knowledge of the nature of the objective function surface, under the problem setting and assumptions outlined here. The 10 test cases are intended as a proxy to multiple forms of objective that might be encountered in a real-world setting. The GP analysis undertaken here would allow for subsequent real-world, expensive to run simulation problems to be tackled using an improved heuristic. It is not the intention that a full GGGP analysis would be applied to an expensive run simulation problem. Furthermore both the general case and individual case GP runs and analysis further enable the assessment of the effectiveness of heuristic sub-components, and their commonality of use across multiple problems. This may be informative in subsequent heuristic development, or in other hyper-heuristic approaches, for example adaptable techniques which can respond to real-time optimisation information and update the search technique employed (Burke et al., 2009, 2013; Pappa et al., 2014).

### 5. Computational experiments

#### 5.1. Experimental set up

The experimental analysis employs 10 established multi-dimensional robust test problems. The problems are listed in Table 1 along with the feasible regions and \( I \)-radius uncertainty values used. For problems taken from the literature on uncertain optimisation, the \( I \)-radius values used are based on their descriptions in Branke (1998), Kruisselbrink et al. (2010, 2011) and Kruisselbrink (2012). For the remaining problems, \( I \)-radius values have been determined by approximating the same relationship between the size of the problem bound, \( X \), and the \( I \)-radius value, as observed for problems in Branke (1998), Kruisselbrink et al. (2010, 2011) and Kruisselbrink (2012). Problem formulations and 2D representations are provided in Appendix A. In our experiments 30D and 100D versions of these problems are considered.

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. There is one GP run for each test problem individually (individual cases) and one for a combined run where each heuristic is applied to all 10 problems (general case). Within each of

| Name                        | \( X \)     | \( I \) |
|-----------------------------|------------|--------|
| Rastrigin                   | [14.88, 25.12] | 0.5    |
| Multipeak F1                | [−5, −4]   | 0.0025 |
| Multipeak F2                | [10, 30]   | 0.5    |
| Branke’s Multipeak          | [−7, −3]   | 0.5    |
| Pickelhaube                 | [−40, −20] | 1      |
| Heaviside Sphere            | [−30, −10] | 1      |
| Sawtooth                    | [−6, −4]   | 0.2    |
| Ackley                      | [17,222, 82,768] | 3      |
| Sphere                      | [15, 25]   | 1      |
| Rosenbrock                  | [7952, 12,048] | 0.25   |

Fig. 5. High level tree-based representation of the heuristic generated by the grammar production rules in Fig. 2, with cut points for combination and mutation operations.

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 Fig. 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 combining 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, (McKay et al., 2010; Miranda and Prudêncio, 2016).
Algorithms are written in Java. For all d.d. calculations the solution of Eq. (11) includes a call to the IBM ILOG CPLEX Optimisation Studio V12.6.3 software.

We now report two analyses. The first, Section 5.2, considers the quality of the best solutions (heuristics) found in the GP runs. The second, Section 5.3, assesses the structure (component breakdowns) of the heuristics generated in the GP runs, against heuristic performance.

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 5.1, are shown in Table 2. Corresponding box plots are shown in Figs. 6 to 9. Individual case results are reasonable, although our use of a much reduced budget for the heuristics developed here should be recognised. Labels on the box plots, indicative only — notwithstanding that the budget of 5000 function evaluations employed by the comparators should somewhat balance the ‘bias’ in these comparisons. The general case comparators were tuned for generality so a direct comparison with our general case results are for the best performing specific heuristic for an individual problem. For the general case the comparator results are for the best performing heuristic overall in Hughes et al. (2020), as applied to all 10 problems.

It should be noted that each heuristic parameter tuning in Hughes et al. (2020) employed a genetic algorithm with population size 20 and 10 generations, compared to each GP run here using population size 500 and six generations. The extent of a GP search is therefore much bigger than a tuning search in Hughes et al. (2020). However the solution space searched in a GP run is substantially larger than the tuning spaces searched in Hughes et al. (2020). Consider, for example, just a handful of the alternative features in the GP analysis which are not in the Hughes et al. (2020) tuning: two PSO velocity formulations, three inner maximisation techniques, three mutation approaches, and seven neighbourhoods. The relative sizes of the spaces searched more than accounts for the population and generation differences between a Hughes et al. (2020) tuning and a GP search here, making the use of comparator results from Hughes et al. (2020) valid.

Comparisons should be interpreted in the context of the use of a budget of 2000 function evaluations for the heuristics developed here. As the individual case comparators were not tuned on specific problems, comparisons with our individual results should be considered indicative only — notwithstanding that the budget of 5000 function evaluations employed by the comparators should somewhat balance the ‘bias’ in these comparisons. The general case comparators were tuned for generality so a direct comparison with our general case results are reasonable, although our use of a much reduced budget for the heuristics developed here should be recognised. Labels on the box plots, Figs. 6 to 9, give the specific comparator heuristic responsible for each set of results.

In Table 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 3D0 in 8 of the individual cases our GP analysis produces the best heuristic, with one worse than the comparator and one
Fig. 6. 30D individual bests box plots. 200 sample runs with a budget of 2000 function evaluations. The comparators are taken from Hughes et al. (2020), where the budget was 5000 evaluations.

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.

For any given problem in Table 2, it can be seen that the general case heuristic typically performs worse than the individual case heuristic, at both 30D and 100D. This is to be expected, as the evolution of a general case heuristic will involve compromising performance for any individual problem in order to achieve reasonable results across all problems. However there are instances where the general case performance is comparable to individual cases, as for the Pickelhaube and Heaviside problems at 30D and Ackley at 100D. It should be appreciated that the extent of the heuristic solution space searched here, evolving a population of 500 heuristics over six generations, is relatively small — limited by practical run time issues, despite the use of parallel processing across multiple cores. It is not surprising, therefore, that some of the GP runs are less successful than others.

The progress of the best performing heuristic in each of the six GP generations is plotted in Appendix B, for each of the 22 GP runs.

5.3. Component analysis

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 3 and 4. The results discussed in Section 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 3 and 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 $r_i$ 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 3 and 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 Fig. 2. For a given component
Fig. 7. 100D individual bests box plots. 200 sample runs with a budget of 2000 function evaluations. The comparators are taken from Hughes et al. (2020), where the budget was 5000 evaluations.

Fig. 8. 30D best general box plots. 200 sample runs with a budget of 2000 function evaluations. The comparators are taken from Hughes et al. (2020), where the budget was 5000 evaluations.
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 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 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 Figs. 10 to 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 Fig. 10 comprising 10 individual test problem plots at 30D, for the type of inner maximisation. There are 3 alternatives for the inner maximisation: random sampling (red), particle swarm optimisation (green), or genetic algorithm (blue). For the Rast-rigin 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 Fig. 10 the proportions of each alternative for the type of inner maximisation is 48%, 24% and 28% for random sampling, PSO and GA respectively. In Sections 5.3.2 to 5.3.11 we consider the main components individually.

5.3.2. Type and extent of inner maximisation

From Table 5 and Figs. 10 to 12 it can be seen that the most used type 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 type 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.

In terms of the number of points evaluated in an inner maximisation, Table 5 and Figs. 13 to 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.

5.3.3. Baseline and extended movement capabilities

Table 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.
Table 5
Proportions of component make ups over all heuristics. Here ‘top’ refers to the top one third of heuristics when sorted best to worst.

| Component                          | 30D all | 30D top | 100D all | 100D top |
|------------------------------------|---------|---------|----------|----------|
| Type of inner search               |         |         |          |          |
| Random                             | 46.6%   | 67.5%   | 42.5%    | 55.3%    |
| PSO                                | 25.2%   | 12.8%   | 29.8%    | 20.5%    |
| 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%    |

Fig. 10. Component — decile breakdowns for the type of inner search, across all GGGP heuristics at 30D. Components: Random (red), PSO (green), GA (blue).

However for the extended form of particle level movement, things are less clear. From Table 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. Figs. 16 to 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.

5.3.4. Network topology

For the form of network for particle information sharing, Table 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, Figs. 19 and 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, Fig. 21, at 30D the Ring (blue) network outperforms Global for the lowest deciles.

5.3.5. Group size

For the group (outer PSO swarm) size, in Table 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 Figs. 22 and 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, Fig. 24, the lower decile results are well distributed across the group size ranges.
5.3.6. Use of a stopping condition

Considering the inclusion of a stopping condition in the inner maximisation, from Table 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.

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 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.

5.3.8. Use of neighbourhood information for $x^*_j$

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.

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 5 and Figs. 25 to 27. A choice of no mutation (red), or mutation due to either Uniform (green) or Gaussian (blue) random sampling, is available. In Table 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.

5.3.11. Extended movement capability features: 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 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.

5.3.12. Extended movement capability features: descent directions $\mathbf{r}_3$ vector

When a heuristic use the descent directions capability (+DD or +DD+LEH), the $\mathbf{r}_3$ vector may be generated randomly or set to the unit vector. Table 5 indicates a reasonably even use of the randomised or unit $\mathbf{r}_3$ 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.

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...
Fig. 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).

Fig. 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).

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 topology, 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.

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 2000 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 Hughes et al. (2019, 2020) and descent directions Bertsimas et al. (2010b), Hughes et al. (2020) 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 Fei et al. (2019), 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.

Alternatives to a \( \Gamma \)-radius description of uncertainty may be considered by adapting some of the sub-algorithmic building blocks developed here. For example by extending from the specialised \( \Gamma \)-radius based...
Marc

CRediT authorship contribution statement

approach, to automatically generate heuristics for robust problems. on moving to the centre of largest empty regions, where a region is
cost points in any uncertainty neighbourhood, or through extending the
descent direction outlined in Bertsimas et al. (2007, 2010a,b) to the
more general concept of a descent direction pointing away from high
cost points in any uncertainty neighbourhood, or through extending the
Γ-radius focussed LEH approach Hughes et al. (2019) to a search based
on moving to the centre of largest empty regions, where a region is
defined by the description of an uncertainty neighbourhood.
A final consideration might be the use of alternatives to the GGGP
approach, to automatically generate heuristics for robust problems.

CRediT authorship contribution statement

Martin Hughes: Conceptualization, Methodology, Software, Formal
analysis, Investigation, Visualization, Writing - original draft. Marc
Goerigk: Supervision, Conceptualization, Methodology, Investigation,
Writing - review & editing. Trivikram Dokka: Supervision, Writing -
review & editing.

Appendix A. 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 Fig. 28. All functions are taken from the literature: Branke
(1998), Kruisselbrink et al. (2010, 2011), Kruisselbrink (2012), Jamil
and Yang (2013).

Rastrigin: \( f(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i) - 10) \)  
\( X = [14.88 \, 25.12]^T \)

Multipeak F1: \( f(x) = -\frac{1}{n} \sum_{i=1}^{n} g(x_i) \)
\( g(x_i) = e^{-5h(\frac{x_i+0.5}{5})} \sqrt{\sin(5\pi x_i + 5)} \) if \( 0.4 < x_i \)
\( e^{-5h(\frac{x_i+0.5}{5})} \sin(5\pi x_i + 5) \) otherwise
\( X = [-5, -5]^T \)

Multipeak F2: \( f(x) = 2 \sin(10 \exp(-0.2(x_1 - 10)(x_2 - 10)) \exp(-0.25(x_3 - 10)) \)
\( X = [10, 20]^T \)

Branke’s Multipeak: \( f(x) = \max\{c_1, c_2\} - \frac{1}{n} \sum_{i=1}^{n} g(x_i) \)
\( g(x_i) = \begin{cases} c_1 \left( 1 - \frac{4(x_i + 0.5)^2}{\eta} \right) & \text{if } 0 \leq (x_i + 5) \leq 2, \\
0 & \text{otherwise} \end{cases} \)
\( b_1 = 2, b_2 = 2, c_1 = 1, c_2 = 1.3 \)  
\( X = [-7, -3]^T \)

Pickelhaube
\( f(x) = \sqrt{5 - \max\{g_5(x), g_6(x), g_7(x), g_8(x)\} \}
\( g_5(x) = \frac{1}{10} \left( 1 - \frac{x + 5}{5} \right) \)
\( g_6(x) = \frac{5}{\sqrt{5}} \left( 1 - \frac{x + 30 + 5}{5\sqrt{2}} \right)^a \)
\( g_7(x) = c_1 \left( 1 - \frac{x + 30 + 5}{5\sqrt{2}} \right)^a \)
\( g_8(x) = c_2 \left( 1 - \frac{x + 30 - 5}{5\sqrt{2}} \right)^a \)
\( c_1 = 625/624, c_2 = 1.5975, d_2 = 1.1513 \)  
\( X = [-40, -20]^T \)

Heaviside Sphere
\( f(x) = 1 - \prod_{i=1}^{n} g(x_i) \right) + \sum_{i=1}^{n} (x_i + 20)^2 \)
\( g(x_i) = \begin{cases} 0 & \text{if } 0 < (x_i + 20) \\
1 & \text{otherwise} \end{cases} \)
\( X = [-30, -10]^T \)

Sawtooth
\( f(x) = 1 - 2 \sum_{i=1}^{n} g(x_i) \)
\( g(x_i) = \begin{cases} (x_i + 5) + 0.8 & \text{if } -0.8 \leq (x_i + 5) < 0.2 \\
0 & \text{otherwise} \end{cases} \)
Fig. 28. Plots of 2D versions of the functions used in our experimental testing.
Fig. 29. Outputs due to the best performing heuristic in each generation of each genetic programme run at 30D. Individual case plots are shown in black whilst general case plots are shown in grey.

Fig. 30. Outputs due to the best performing heuristic in each generation of each genetic programme run at 100D. Individual case plots are shown in black whilst general case plots are shown in grey.
Appendix B. Progress of best performing heuristics

Progress of the best performing heuristic out of the population of 500, over each of the six generations, is shown in Figs. 29 and 30 — for each of the 20 individual GP runs and each of the two general GP runs (which are applied to all 10 test problems). Values shown are estimated robust global minimum values, calculated as the mean of 20 sample runs of the same heuristic applied to the same problem.

References

Bader, M., Poli, R., Fatima, S., 2009. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. Memetic Comput. 1 (3), 205–219, Funders: EPSRC.

Ben-Tal, A., Nemirovski, A., 2009. Robust Optimization. Princeton University Press, Princeton and Oxford.

Ben-Tal, A., Nemirovski, A., 1998. Robust convex optimization. Math. Oper. Res. 23 (4), 769–805.

Bertsimas, D., Brown, D.B., Caramanis, C., 2011. Theory and applications of robust optimization problems with constraints. INFORMS J. Comput. 22 (1), 44–58.

Bertsimas, D., Nohadani, O., Teo, K.M., 2010. Parallel Problem Solving from Nature. PPSN XI, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 119–128.

Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R., 2013. Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. 64 (12), 1695–1734.

Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R., 2009. Exploring hyper-heuristic methodologies with genetic programming. In: Computational Intelligence: Collaboration, Fusion and Emergence. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 177–201.

de Carvalho, D.F., Bastos-Filho, C.I.A., 2009. Clon particle swarm optimization. Int. J. Intell. Comput. Cybern. 2 (2), 197.

Chen, R., Lucier, B., Singer, Y., Syrgkanis, V., 2017. Robust optimization for unconstrained simulation-based problems. Oper. Res. 58 (1), 161–178.

Chen, R., Lucier, B., Singer, Y., Syrgkanis, V., 2017. Robust optimization for unconstrained simulation-based problems. Oper. Res. 58 (1), 161–178.

Chung, W., Yu, M., 2013. Robust Optimization for Non-convex Optimization Problems. pp. 141–142.

Engelbrecht, A., 2012. Particle Swarm Optimization: Velocity Initialization. In: 2012 IEEE Congress on Evolutionary Computation, pp. 1–8.

Fei, X., Branke, J., Gülpinar, N., 2019. New sampling strategies when searching for robust solutions. IEEE Trans. Evol. Comput. 23 (2), 273–287.

Gabriel, V., Murat, C., Thiele, A., 2014. Recent advances in robust optimization: An overview. European J. Oper. Res. 235 (3), 471–483.

Goerigk, M., Hughes, M., 2019. Representative scenario construction and preprocessing for robust combinatorial optimization problems. Optim. Lett. 13 (6), 1417–1431.

Goerigk, M., Schöbel, A., 2016. Algorithm engineering in robust optimization. In: Kliemann, L., Sanders, P. (Eds.), Algorithm Engineering: Selected Results and Surveys. In: Lecture Notes in Computer Science, vol. 9220 of LNCs State of the Art, Springer Berlin / Heidelberg, pp. 245–279.

Herrmann, J.W., 1999. A genetic algorithm for minimax optimization problems. In: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, Vol. 2. IEEE, pp. 1099–1103.

Hughes, M., Goerigk, M., Dokka, T., 2020. Particle swarm metaheuristics for robust optimization with implementation uncertainty. Comput. Oper. Res. 122.

Hughes, M., Goerigk, M., Wright, M., 2019. A largest empty hypersphere metaheuristic for robust optimisation with implementation uncertainty. Comput. Oper. Res. 103, 64–80.

Jančević, D., Jelenković, L., Budin, L., 2007. Genetic programming heuristics for multiple machine scheduling. In: Ehner, M., O’Neill, M., Ektö, A., Vanneschi, L., Esparcia-Alcázar, A.I. (Eds.), Genetic Programming. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 321–330.

Janži, M., Yang, X.-S., 2013. A literature survey of benchmark functions for global optimization problems. Int. J. Math. Modell. Numer. Optim. 4 (2), 150–194.

Janson, S., Middendorf, M., 2005. A hierarchical particle swarm optimizer and its adaptive variant. IEEE Trans. Syst. Man Cybern. B 35 (6), 1272–1282.

Jansen, M.T., 2004. A new look at solving minimax problems with coevolutionary genetic algorithms. In: Metaheuristics: Computer Decision-Making. Springer US, Boston, MA, pp. 369–384.

Kameyama, K., 2009. Particle swarm optimization - A survey. IEICE Trans. Inf. Syst. E92.D (7), 1534–1561.

Kennedy, J., Eberhart, R.C., 1995. Particle swarm optimization. In: Proceedings of the 1995 IEEE International Conference on Neural Networks, Vol. 4, Perh, Australia. pp. 1942–1948.

Kennedy, J., Eberhart, R.C., Shi, Y., 2001. Swarm Intelligence. Morgan Kaufmann.

Kennedy, J., Mendes, R., 2002. Population structure and particle swarm performance. In: Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No.02TH8600), Vol. 2. pp. 1671–1676.

Kimura, M.S., 2017. Particle swarm optimization with a new update mechanism. Appl. Soft Comput. 60, 670–678.

Kouvelis, P., Yu, G., 1997. Robust Discrete Optimization and Its Applications. Kluer Academic Publishers.

Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. In: A Bradford book, MIT Press.

Kruiselbrink, J.W., 2012. Evolution Strategies for Robust Optimization (Ph.D. thesis). Leiden Institute of Advanced Computer Science (LIACS), Faculty of Science, Leiden University.

Kruiselbrink, J., Emmerich, M., Bäck, T., 2010. An archive maintenance scheme for constrained optimization problems by using a modified particle swarm optimization. In: Systems, Man, and Cybernetics, 2010 IEEE International Conference on, Vol. 2. IEEE, pp. 1099–1103.

Kruiselbrink, J.W., Rheeui, E., Deutz, A., Bäck, T., Emmerich, M., 2011. Using the uncertainty handling CMA-ES for finding robust optima. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. GECCO ’11, ACM, New York, NY, USA, pp. 877–884.

Kusiak, A., 1997. Robust optimization with implementation uncertainty. Comput. Oper. Res. 122.

van Lon, R.R.S., Holvoet, T., Van den Berge, G., Wenseleers, T., Branker, J., 2012. 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, Association for Computing Machinery, New York, NY, USA, pp. 331–336.

Marat, Z., Walter, E., Piet-Lahanier, H., 2013. Worst-case global optimization of black-box functions through Kriging and relaxation. J. Global Optim. 55 (4), 707–727.

Marat, Z., Walter, E., Piet-Lahanier, H., 2016. A new expected-improvement algorithm for continuous minimax optimization. J. Global Optim. 64 (4), 785–802.

Masch, Franco, López-Ibáñez, Manuel, Dubois-Lacoste, Jérémie, Stützle, Thomas, 2014. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. Comput. Oper. Res. 51, 190–199.

Masuda, K., Kuribara, K., Aiyoshi, E., 2011. 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. IEEE, pp. 2113–2120.

Mckay, R.L., Hoai, N.X., Whigham, P.A., Shan, Y., O’Neill, M., 2010. Grammar-based genetic programming: a survey. Genet. Programming Evol. Mach. 11 (3), 365–396.

Mei, Y., Zhang, M., 2018. Genetic Programming Hyper-Heuristic for Multi-Vehicle Uncertain Capacitated Arc Routing problem. pp. 141–142.

Mello, T., Homem, Bayraksan, G., 2014. Monte Carlo sampling-based methods for robust optimization with implementation uncertainty. Surv. Oper. Res. Manag. Sci. 19 (1), 56–85.

Mendes, R., Kennedy, J., Neves, J., 2003. 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), pp. 88–94.
Miranda, P.B.C., Prudêncio, R.B.C., 2016. Tree-based grammar genetic programming to evolve particle swarm algorithms. In: 2016 5th Brazilian Conference on Intelligent Systems. BRACIS, pp. 25–30.

Miranda, Péricles B.C., Prudêncio, Ricardo B.C., 2017. Generation of particle swarm optimization algorithms: An experimental study using grammar-guided genetic programming. Appl. Soft Comput. 60, 281–296.

Mirjalili, S., Lewis, A., Mostaghim, S., 2015. Confidence measure: A novel metric for robust meta-heuristic optimisation algorithms. Inform. Sci. 317, 114–142.

Nickabadi, A., Mehdizadeh, M., Safabakhsh, R., 2011. A novel particle swarm optimization algorithm with adaptive inertia weight. Appl. Soft Comput. 11 (4), 3658–3670.

Nohejl, A., 2011. Grammar-Based Genetic Programming (Master’s thesis). Department of Software and Computer Science Education, Faculty of Mathematics and Physics, Charles University in Prague.

Ong, Y.-S., Nair, P.B., Lim, K.Y., 2006. Max-min surrogate-assisted evolutionary algorithm for robust design. IEEE Trans. Evol. Comput. 10 (4), 392–404.

Paenke, I., Branke, J., Jin, Y., 2006. Efficient search for robust solutions by means of evolutionary algorithms and fitness approximation. IEEE Trans. Evol. Comput. 10 (4), 405–420.

Pappa, G.L., Freitas, A.A., 2010. Automating the Design of Data Mining Algorithms: an Evolutionary Computation Approach. In: Natural Computing Series, Springer, pp. 182–196.

Pappa, G.L., Ochoa, G., Hyde, M.R., Freitas, A.A., Woodward, J., Swan, J., 2014. Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. Genet. Programming Evol. Mach. 15 (1), 3–35.

Poli, R., Di Chio, C., Langdon, W.B., 2005. Exploring extended particle swarms: A genetic programming approach. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation. GECCO ’05, ACM, NY, USA, pp. 169–176.

ur Rehman, S., Langelaar, M., 2017. Expected improvement based infill sampling for global robust optimization of constrained problems. Opt. Eng. 18 (3), 723–753.

ur Rehman, S., Langelaar, M., van Keulen, F., 2014. Efficient Kriging-based robust optimization of unconstrained problems. J. Comput. Sci. 5 (6), 872–881.

Robinson, J., Rahmat-Samii, Y., 2004. Particle swarm optimization in electromagnetics. IEEE Trans. Antennas and Propagation 52 (2), 397–407.

Sanders, N.D., Eversen, R.M., Fieldsend, J.E., Rahat, A.A.M., 2019. A Bayesian approach for the robust optimisation of expensive-to-evaluate functions. arXiv e-prints, arXiv: 1904.11416.

Sengupta, S., Basak, S., Peters, R., 2018. Particle Swarm Optimization: A survey of historical and recent developments with hybridization perspectives. Mach. Learn. Knowl. Extract. 1 (1), 157–191.

Shi, Y., Eberhart, R., 1998. A modified particle swarm optimizer. In: 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360). pp. 69–73.

Shi, Y., Krohling, R., 2002. Co-evolutionary particle swarm optimization to solve minimax problems. In: Evolutionary Computation, 2002. CEC’02. Proceedings of the 2002 Congress on, Vol. 2. IEEE, pp. 1682–1687.

Stephenson, M., Amarasinghe, S., Martin, M., O’Reilly, U.-M., 2003. Meta optimization: Improving compiler heuristics with machine learning. SIGPLAN Not. 38 (5), 77–90.

Tamh, E.-G., 2009. Metaheuristics: From Design to Implementation. Wiley Publishing.

Tsubui, S., Ghosh, A., 1997. Genetic algorithms with a robust solution searching scheme. IEEE Trans. Evol. Comput. 1 (3), 201–208.

Vu, K. Khac, D’Ambrosio, C., Hamadi, Y., Liberti, L., 2016. Surrogate-based methods for black-box optimization. Int. Trans. Oper. Res. 24 (3), 393–424.

Wang, L., Yang, B., Orchard, J., 2016. Particle swarm optimization using dynamic tournament topology. Appl. Soft Comput. 48 (C), 584–596.

Whigham, P.A., 1995. Grammatically-based genetic programming. In: Proceedings of the Workshop on Genetic Programming: From Theory To Real-World Applications. University of Rochester, USA, pp. 33–41.

Zhang, X., Wang, S., Ji, G., 2015. A comprehensive survey on particle swarm optimization algorithm and its applications. Math. Probl. Eng. 2015.

Zhou, A., Zhang, Q., 2010. A surrogate-assisted evolutionary algorithm for minimax optimization. In: IEEE Congress on Evolutionary Computation, pp. 1–7.