Momentum and Branching: a Promising Global Optimisation Strategy for Sparse Problems

Abstract

Non-convex discrete multi-objective black-box optimisation problems such as feature selection for AI models represent a growing class of optimisation problems. Few generalised algorithms exist to solve these specific optimisation problems due to their complex and sparse nature, yet due to their relevance to modern day applications, there is a growing need for general optimisation strategies in order to consistently solve these problems types. This paper designs and implements a generalised optimisation algorithm combining a branching heuristic to exploit local minima, as well as a novel momentum heuristic in order to reduce redundant computations and increase exploration. The algorithm further extends modern reinforcement learning frameworks in order to solve non-convex discrete multi-objective black-box optimisation problems and provides an alternative to multi-agent solutions for reinforcement learning's viability in optimisation. The complete system is compared to popular optimisation algorithms on a mixture of black-box travelling salesman problems and demonstrates the best performance in small scale problems, as well as better performance compared to similar algorithms such as epsilon-greedy reinforcement learning on all problem instances. The use of novel branching and momentum heuristics also show great potential as well as extendibility to modern meta-heuristic solutions but require careful parameter selection in order to be effective.

Contents

1 Introduction

Optimisation is a wide field used throughout the engineering sciences. From optimising features in artificial intelligence models, to refining business strategies in economics, and advancing robot automation, optimisation is central to the efficacy of various businesses; where the key to solve challenges faced in these industries is by breakthroughs in optimisation techniques[1] [2]. Modern industries continually face larger and more demanding problems, which even state-of-the-art optimisation algorithms are unable to tractably solve, pushing for the continuous development of larger, more capable specialist algorithms [3]. This race for the most competitive optimisation has created an environment in which many micro-enterprises and small businesses struggle to employ specialist systems without incurring significant running costs to enable them to keep up with competition [4] [5].

In practice, optimisation algorithms are usually employed to tackle problems classified as NP-hard (non-deterministic polynomial-hard). NP-hard problems are a subset of problems in P, those computable in polynomial time. For example matrix-matrix multiplication, where increasing the size of the matrix by n, leads to n^3 more calculations necessary for a naive algorithm. NP problems differentiate themselves from problems solvable in P time as they are non-deterministic, in other words no algorithm is proven to solve NP problems in polynomial time [6]. NP-hard problems have the further challenge that no algorithm exists to check whether a given solution is the best one. As such NP-hard problems entail the edge of computability, where the domain encompassing valid solutions to problems is too large to employ a brute-force strategy, a deterministic approach which takes exponentially longer with an increase in size, but no deterministic algorithm exists to solve it in P time [7]. As a consequence optimisation algorithms are used to approximate answers in P time by using heuristics to guide their search towards good solutions without the need to test every possible solution. As the time to compute grows rapidly with size for problems in P, NP-hard problems over a certain magnitude are still intractable nowadays and justifies the continuous need for improvement in optimisation algorithms. But this is no easy task, optimisation algorithms must balance both exploration, to decrease the chance a solution settles on a local minima, and exploitation, to converge on a good solution in P time [8].

Non-convex multi-objective discrete black box optimisation problems are a notoriously hard subset of optimisation problems (OPs) where general algorithms perform much worse than expert-algorithms tailored to specific problems. Furthermore, recent research shows that popular state-of-the-art multi-objective evolutionary algorithms (MOEAs) are inconsistent, clustering solutions to different local optima in various executions, showing behaviour similar to local search methods when tackling non-convex discrete multi-objective OPs [9]. This suggests the need for new algorithms more capable at handling global optimisation in non-convex multi-objective discrete OPs. In addition to this, the recent surge in AI popularity has driven the need for black box optimisation, especially in discrete cases where problems such as feature selection are more relevant than ever. In order to increase the accessibility of AI model development to non-expert audiences, performant, consistent and easy to implement optimisation algorithms are more necessary than ever. This constitutes the primary motivation for this project.

In section 1.1 a brief definition of non-convex discrete multi-objective black-box optimisation problems is given. Following this, a literature review presenting an overview of relevant research is provided, before outlining the project goals. Section 3 will present a high level algorithm design outlining import aspects considered in order to meet the desired requirements. The design is followed by the production log outlined in Section 4, which discusses the development made to the algorithm, along with any additions and changes made in face of challenges. Thereafter, results will be discussed in section 5, comparing the designed system's performance to state-of-the-art algorithm. The report is concluded by critically evaluating the suggested algorithm to the initial goals and discussing future work relevant to the suggested algorithm.

1.1 Background Information

Optimisation is a systematic approach used for minimising or maximising (optimising) the value of an objective function with respect to a set of constraints [10]. We can formalise this as a problem under the form:

$$\begin{aligned} &\text{opt. } f(x) \\ &\text{s.t. } x \subseteq S \end{aligned} \tag{1}$$

Where $f : R^n \rightarrow R$ is the objective function, x the decision variable and $S \subseteq \mathbb{R}^n$ the feasible set of values for x. The abbreviations opt. and s.t. are short for *optimising* (minimising or maximising) and *subject to* respectively. The feasible set of values S for x is a subset of $\mathbb{R}^n$, and hence $x = (x_1, x_2, ..., x_n)$ is the vector of variables of dimension n and f is a function of n real values $f(x_1, x_2, ..., x_n)$.

Optimisation remains a huge field with many specialty branches employing different strategies due to certain limitations. Holistically all contemporary optimisation problems have domains too large to brute force, either by being a high-dimensional problem, or the range for each dimension being very large. Hence heuristics are used to determine where to sample the objective function in order to find the optima. As such the main premise for all algorithms revolves around exploration and exploitation, using as much information as possible in order to exploit promising regions more extensively and explore enough to be confident the optima has been found. Different OPs are all structured in different ways, which consequently allow for different information to be used in algorithms designed to tackle them, but can be generalised into types. This project will focus on non-convex discrete multi-objective black box OPs, a subset of optimisation with many real world problems such as DNA binding or black-box scheduling [11] [12].

- **Non-Convex -** Convexity refers to the shape of the objective over the domain of possible values to be evaluated, where a function $f(x)$ is convex if it satisfies Jensen's inequality, which states "the secant line lies above the graph of $f(x)$" for any $x_1, x_2 \in x$. For OPs where the objective function is convex this then has the property: "any locally optimal point is globally optimal. In addition, the optimal set is convex." [13]. OPs which are proven to be convex can then easily be solved using any gradient descent or hill-climber algorithm, algorithms which compare nearby points and explore in any direction where immediate solutions are better than the current one. Non-convex problems on the other hand can have multiple local optima which are not the global optima, in these problems gradient descent or hill-climbers are far less effective as they are likely to get trapped in a local optima, missing the best possible solution.

- **Discrete -** Continuous and discrete OPs refer to the domain of the objective function being optimised. For a continuous function, the valid domain is any number within the allowed range, whereas in discrete problems the allowed domain is a set of predefined values. Many discrete problems have domains where their sets have little to no associated structure, for example in 0-1 NK landscapes the feasible set for N dimensions is $S = \{0, 1\}^n$, representing on or off for each dimension respectively. As a consequence discrete problems are unable to use gradient information as a set of values with no associated structure has no derivative unlike a continuous domain. This is a large detriment for discrete algorithms as gradient information can be very effective at guiding search efforts, such as the use of Hessian matrices in CMA-ES.

- **Multi-Objective -** OPs seek to optimise a given objective function, in multi-objective OPs a proposed solution is graded on multiple typically competing objective functions. The given set of objective functions must all be optimised and no objective is prioritised over any other, therefore heuristics in this case try to find sets of pareto-optimal solutions, where each pareto-optimal solution in the final set is better than every other pareto-optimal solution in at least one objective. Multi-objective optimisation algorithms tend to prioritise more thorough exploration in order to find a good pareto-optimal set of solutions, this can be challenging for greedier algorithms as well as those which focus on fine-tuning a single solution.

- **Black-Box -** In Black box problems, no additional information is given to a solver other than how well a solution scores when testing the objective function. This gives an additional challenge to solvers as they

must deduce any local information themselves. As a consequence greedy strategies are substantially less reliable as local information deduced by exploration and surrogate models is always prone to error.

2 Literature Review and Project Specification

The numerous constraints imposed on non-convex discrete multi-objective black-box OPs creates an environment where high-performing generalised optimisation algorithms such as popular MOEA non-dominated sorting genetic algorithm (NSGA-II) are unable to perform reliably [9]. In practice research showed they performed similar to local optimisation algorithms not reaching their global optimisation goals. It is well known that no one optimisation algorithm can conquer all, and under different constraints performance is very variable. On the other hand, it is found that certain general algorithms perform very well in specific niches, where they reliably solve many similar OPs with good performance without the need for modification [14]. As such this creates a research opportunity to find general reliable optimisation algorithms for non-convex discrete multi-objective black box OPs.

The travelling salesman problem (TSP) is a classical non-convex discrete constrained OP. Given a set of cities, the goal of the TSP is to find the shortest tour joining every city to each other, with the constraints that every city must be visited, and only visited once. In the TSP we can consider the cities to be the dimensions, so a TSP of n cities has n dimensions, where a solution vector to the objective function minimising the TSP tour represents the order in which the cities are connected. The TSP is a well studied problem, with many variations allowing for applications to scheduling, microchip manufacturing and even genome sequencing [15]. The impact solving TSP like problems has led to many specialist algorithms, with modern deterministic algorithms designed in order to get optimal solutions. Notably algorithms such as branch and bound (B&B), and branch and cut (B&C) were implemented to solve it deterministically [16]. B&C is a specialist algorithm designed for constrained problems, it initially solves the problem without constraints, in the case of the TSP where the constraints specify that every city can only be visited once enabling a solution which doesn't adhere to this. It then adds cutting planes which force the solution to converge on a local constrained solution, for example forcing an unconstrained solution going through the same city multiple times to cut a transition for the city visited multiple times. B&B is designed around non-convex discrete unconstrained, but white box problems. It works by creating a solution tree, where the root represents a single dimensional value, in other words the root city. Branches are created for every possible valid transition away from this city, and then bounds are calculated to estimate the best and worst case solution deliverable from this branch. This process is repeated for each branch node until a leaf node is reached, representing a full valid tour following the transitions described by the branches above it. In the worst case this tree-exploration has an exponential run time, as it allows for every solution permutation to be considered. But the calculated bounds at each branch allow for the algorithm to deterministically ignore branches which are shown to have no potential. Depending on the bounding algorithm as well as the problem instance this allows B&B to perform in polynomial time whilst finding the optima deterministically [17].

B&B and B&C are interesting alternatives to heuristic solutions, the benefits of zero redundant computations unlike many meta-heuristic algorithms as well as finding optima in non-convex landscapes makes them very popular and well researched. Furthermore non-deterministic variations are often used in very high dimensional problems, where high level heuristics can leverage the branching, bounding and cutting properties whilst maintaining a worst case polynomial run time but also accepting non-optimal solutions. B&B and B&C are a great example of algorithms which effectively leverage local information known in white-box OPs. In black-box problems these algorithms struggle to perform as reliably due to expensive surrogate models which are needed to model the local information along the objective function in order to allow these algorithms to effectively bound and cut [18]. Unfortunately surrogate models are never completely accurate therefore the bounds and cuts and often take a lot of computation time in order to build accurately.

Evolutionary algorithms (EAs) encompass a very popular family of population-based meta-heuristics. These

predominantly work through the use of selection pressure; where each generation the population of solutions culled, or ranked, before repopulating the population for the next generation. Classically genetic algorithms (GA) are very well researched, with popular algorithms such as NSGA-II (non-dominated sorting GA-II) widely used for multi-objective discrete optimisation. GAs are very easy to implement and paved the way for early efforts in optimisation algorithms, the main premise of selection, mutation and crossover is very generalisable but many problem-specific variants have been invented in order to address weaknesses in GA. Predominantly, GA is cursed to try to maintain the genetic diversity of its population to allow for thorough exploration, directly slowing down the efforts to exploit promising solutions in its population [19]. Maintaining good genetic diversity to enable good exploration whilst allowing for efficient exploitation of promising solutions is in itself a very hard problem [20]. Furthermore choosing the best heuristic for selection, mutation and crossover is typically very problem-specific.

Ant-colony optimisation (ACO) is another EA which is based around a population of ants laying down pheromone trails on visited promising solutions in order to guide the decision making of future ants towards good paths. Unlike in GA, ACO uses a probability matrix calculated by a mixed heuristic function combining local and learnt global information. This probability matrix is used by virtual ants to choose which transition to take in order to traverse through a full solution efficiently. This iterative process of generating solutions is similar to brand and bound, where a single dimension is considered initially, before iteratively building up a solution until it represents a valid solution. Figure 1 below represents a 4 city TSP instance, table 1 adjacent to it shows a matrix representation for the TSP problem where the rows and columns represent each city, and the values in the matrix represent the path length between each city, note that transitions from a city to itself is noted as 0 and is not an allowed transition. This matrix can be used as a greedy transition matrix, representing local information, where the shortest path for each city is represented by the smallest number in each column. We could then imagine a greedy virtual ant starting at city A, choosing its next destination using this transition matrix as $A \rightarrow B$, with the shortest path length of 20. Next looking at row B, the next shortest transition is $B \rightarrow C$, as the transition $B \rightarrow A$ is no longer allowed as A is now visited. Finally, the ant arrives at city C, and looking at row C the only allowed transition is $C \rightarrow D$. Following this logic We can see that using a greedy search the ant picks the final path $A \rightarrow B \rightarrow C \rightarrow D(\rightarrow A)$, which happens to be the optima. Its interesting to note that using a probability matrix as this to generate a solution, we can describe the solution as a series of transitions (or edges for the fully connected graph case) rather than the individual nodes. In practice there is also a stochastic element to the ant decision making allowing for exploration, as well as a second heuristic called pheromone which represents global information, allowing the ant to balance local greedy information with global information.

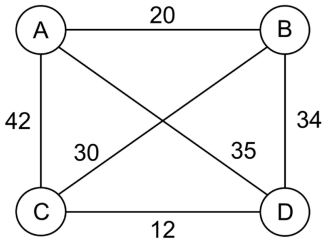


Figure 1: Graph representation of 4 city TSP, taken from [21]

	A	B	C	D
A	0	20	42	35
B	20	0	30	34
C	42	30	0	12
D	35	34	12	0

Table 1: Transition matrix for adjacent 4 city TSP problem

Using a probability matrix to generate solutions means ACO does not struggle to maintain genetic diversity like in GA. As the probability matrix can cover all possible transitions, its easy to add functions to induce exploration [22]. ACO is very popular for non-convex discrete white box optimisation, but struggles with black-box problems as local information, used to influence the ant decision-making, is not given. Furthermore, ACO has the additional challenge of balancing its population size, how strongly pheromone evaporates and how much local vs global information influences the calculated transition probabilities [23].

Reinforcement learning (RL) is a prominent modern machine learning technique, with recent successes such as AlphaGo it has proven itself to be a useful tool for decision-making in large decision spaces [24]. In fixed

optimisation problems, reinforcement learning has seen less use as it is largely outperformed by meta-heuristic algorithms. The static environment of OPs lends itself well to meta-heuristic algorithms which can quickly find an optima, whereas RL has a longer training period used to build more concrete policies in order to get consistent good results. Despite this RL methods promise easy-to-implement solutions, where efficient RL algorithms will be easily generalisable to any problem. RL algorithms would also be well prepared for black box problems, where its reward mechanism can be effectively used with no local information [25]. Furthermore promising research is found when merging deep-learning with RL decision making. Although individual optimisation problems were solved more optimally by traditional meta-heuristic algorithms, in a dynamic problem environment the creation of a almost-optimal policy by a deep RL algorithm allowed for almost instantaneous solving for dynamically changing problems [26].

Fundamentally RL is also limited under the Markov assumption they work under. The Markov property expresses that the likelihood of changing to a specific state is reliant exclusively on the present state and elapsed time and not on the series of states that have preceded it [27]. As a consequence, context-free RL agents will only converge on a single optima. This leaves them susceptible to non-convex problems, as well as multi-objective problems where the goal is to find sets of pareto-optimal solutions. Approaches such as multi-agent RL can be used to account for this but requires extra overhead in order to work efficiently [28].

The many limitations imposed on non-convex discrete multi-objective black-box problems mean many popular and effective optimisation methods mentioned above struggle. Algorithms such as B&B or ACO struggle due to the lack of local information, in addition methods such as RL are yet to show performance on par with the above mentioned meta-heuristic algorithms. Furthermore, Viable algorithms such as GA, a type of MOEA, have been shown to be ineffective at global optimisation and behave similarly to local random search algorithms [9]. As such this project aims to create a consistent optimisation algorithm, extending on exisiting research in order to offer a consistent framework for solving non-convex discrte multi-objective black-box OPs. This is an ambitious task as the constraints and lack of information in these OPs may require algorithms to be hand-made to solve them, in order to leverage problem-specific information as seen in the current trend of problem-specific surrogate-model assisted algorithms when dealing with these OPs.

2.1 Requirements

- **Non-Convex Discrete Black-Box Optimisation -** A proposed algorithm must be able to effectively work under the constraints of non-convex discrete black-box optimisation. As such it must be designed around methods ignoring local information as well as gradient information. Furthermore it should be built under the assumption of multiple local minima, where the final algorithm is able to explore the solution space in order to increase certainty of finding the optima. Ideally information found in non-convex discrete black-box problems should be leveraged.

- **Multi-objective Capabilities -** Multi-objective design paradigms must not be an afterthought. Consequently, methods which forcibly search for a single optima should be ignored. The final solution should be able to leverage modern algorithms such as non-dominated sorting or be able to store a set of good solutions in order to comply with this requirement. Ideally the convergence mechanism of the proposed algorithm will be able to simultaneously explore and converge on multiple optima without difficulty.

- **Generalised -** The algorithm should be a general solution to non-convex discrete multi-objective black box optimisation problems. As such it should not be built around problem specific features but general features found in this subclass of problems. As a consequence the algorithm should be able to solve different problems with little to no modifications apart from adding constraints for constrained OPs.

- **Extendability -** To comply with modern research the proposed algorithm should be built around existing paradigms. This would allow problem-specific research to be easily extendable to the proposed solution or at

least shed light onto the feasibility of the proposed modifications to any modern algorithms used. Preferably the proposed algorithm will be able to be used by up and coming methods such as deep learning or RL, or concepts able to be drawn over to popular meta-heuristic approaches such as GA or ACO.

- **Polynomial Time Optimisation -** The proposed heuristic should be able to find near-optimal solutions in Polynomial time in order to be competitive with other meta-heuristic algorithms. Ideally the algorithm will find solutions at least as quickly as general meta-heuristic approaches whilst maintaining similar solution quality. If possible the proposed solution will have both better convergence capabilities enabling better solutions to be found whilst maintaining the same or better time complexity as meta-heuristic algorithms.

2.2 Evaluation

- **Testing on Diverse Problems -** Previous research notes the importance of using a diverse problem set when benchmarking algorithms [29]. This is important to ensure that the proposed solution is a good general solution for non-convex discrete black-box OPs rather than being optimised for a specific problem. A diverse benchmarking set can identify weaknesses in the proposed algorithm as well as good use cases for the algorithm.

- **Consistent Performance** State-of-the-art optimisation algorithms have been shown to perform similarly to local search algorithms when optimising non-convex discrete multi-objective OPs[9]. Consequently, if the proposed algorithm is able to consistently find the same or similar solutions, this will show its potential for global optimisation in non-convex discrete multi-objective search spaces. This can be shown by repeatedly running the algorithm and comparing results along multiple executions.

- **Performance Compared to State-of-the-art Algorithms** In order to be a competitive option the suggested algorithm should have similar run time to other widely-used meta-heuristic algorithms. This further measured into having thorough exploration whilst maintaining thorough exploitation. A further comparison of solution quality as well as diversity in solution sets to state-of-the-art algorithms will be important to see if the proposed algorithm is able to adequately find optima.

3 Design

3.1 High-Level Algorithm Design

Optimisation algorithms in general work by refining a solution candidate or solution generator in order to converge on an optimal solution. We can break this process down into four primary elements: initialisation, generation, selection, and a fourth element which controls these three aforementioned parameters [30]. Algorithm 3.1 demonstrates the above key components and structure for generalised optimisation algorithms (as seen in fig 2) [31]. The four key elements must fundamentally balance exploratory efforts along the objective function's domain in order to try find the region containing the optima, while exploiting any promising regions in order to locate the optima within regions. The following section will outline how these four key elements can be incorporated when solving non-convex discrete multi-objective black box optimisation whilst adhering to the requirements outlined in 2.1.

- **Initialisation -** In black box optimisation the initial spread of the population has a great impact on further exploration [32]. Having a wide distribution for the initial exploration is paramount, where methods such as Halton sampling can be used in continuous landscapes [32]. To ensure a wide initial spread in a discrete landscape, space-filling designs such as a population based algorithm could be implemented in order to get an initial exploration of the solution landscape. Methods such as multi-start, locally optimising a number of initial candidates in order to get an initial spread of local optima, are less effective in discrete black-box environments where gradient information is unavailable and local optimisation algorithms are less efficient.

- **Generation -** After the initialisation step, a choice must be made for how future generations are instantiated. GA offer an efficient method for creating offspring, especially in constrained problems where the right crossover heuristic can efficiently create children combining the information of two parent solutions. On the other hand research shows that probabilistic methods for solution generation outperform crossover methods [33]. Probabilistic methods are used in algorithms such as ACO which use a probability matrix to guide the ants as they solve the problem, or in RL based algorithms, where the Q matrix defining possible transitions can be comparable to a probability matrix. Probabilistic methods are well suited for discrete problems, where the finite feasible set S describing possible states for each dimension can be easily transposed into a probability matrix. Using the right heuristic for calculating the probability matrix will be paramount to ensure the future solutions are able to exploit regions of interest and explore unseen regions. Its worth mentioning that due to the black-box nature of the problems this paper looks at, using a deterministic strategy for solution generation as seen in B&B and B&C would lead to worst case exponential time complexity, without the benefits of efficient bounding or cutting due to no local information, as such a Probabilistic method for solution generation seems the most viable option. Furthermore, using a probabilistic method for generation would allow extendability for research in RL optimisation algorithms as well as EAs such as ACO in line with the requirements stated in section 2.1.

- **Selection -** The selection step is used to select the best individuals within each generation, Typically used for population based algorithms it creates selection pressure to propagate the better individuals. It is also synonymous with solution ranking in non-population based algorithms, allowing a qualitative analysis of how good a solution is relative to global search efforts. When using a population based algorithm as suggested in the initialisation step, many strategies for selection can be easily implemented by comparing the solution quality of each candidate within the population. Thereafter, by ranking the solutions they can influence the transitions they took in the probability matrix, with better solutions influencing the probability matrix more. Otherwise, the score each solution achieves can be used directly as their rank, where a system similar to Q-learning from RL can be implemented to update the probability matrix. In section 3.3 a branching mechanism will be introduced in which selection criteria will also be required.

- **Control -** Using a population and probability matrix as mentioned above, the control element will decide how best to update the probability matrix in between iterations. This will include methods such as evaporation as used for pheromone trails in ACO, as well as using an offset to encourage exploration. Furthermore a momentum parameter will be introduced in section 3.4 which will heavily influence how the probability matrix is calculated. Finally there will be external control parameters in the final algorithm, these will have

Algorithm 3.1: General Optimization Algorithm

```
1  set initial control parameters
2  begin
3      t = 0
4      initialize candidate(s)
5      evaluate initial candidate(s)
6      while not termination-condition do
7          t = t + 1
8          generate new candidate(s)
9          evaluate new candidate(s)
10         select solution(s) for next iteration
11         optional: update control parameters
12     end
13 end
```

Figure 2: Fundamental structure for optimisation algorithms taken from [31]

to be adjusted by the user prior to the optimisation run. This will include parameters such as number of iterations or population size.

The solution landscape presented by non-convex discrete multi-objective black-box OPs is quite unique in that very little to no information is available to the user. As such methods using gradient information along each dimension or local information are infeasible. In the following subsection a brief discussion of what information can be found in these problem types, and how that can influence the design of a general optimisation algorithm for this problem type.

3.2 Information in Discrete Black-Box Environments

As discussed in section 1.1, discrete black-box OPs are very limited in the information they contain. In order to extract information to be used by a heuristic a primary assumption has to be made, namely that the objective function is not fully noisy. In other words the objective function has some observable trend, and by sampling a point, this reveals information about nearby points due to some unknown correlation. This property allows for optimisation to occur in general, but has also motivated a large area of optimisation research namely the use of surrogate-models. Surrogate models use methods such as gaussian processes (as seen in Bayesian optimisation strategies), radial-basis-function interpolation, or even deep neural networks in order to simulate the objective function along its domain by sampling a distribution of points in the objective function [31] [30]. Surrogate models have shown to be very good in Black-box environments, especially those where function calls are expensive, where good models allow for strategic sampling in order to model the objective function as efficiently as possible. Unfortunately, surrogate models need to be combined with other meta-heuristic approaches, research shows that models created from a number of samples will never be able to show full information regarding minima and optima, requiring heuristics to aid the search in order to find these in promising regions [34]. Furthermore surrogate-models are very problem specific, where different methods of creating the surrogate models are better suited for different problems, especially in discrete OPs where there is no guarantee of associated structure within the feasible set of values for each dimension [35]. Therefore in the endeavour to find a suitable general optimiser a model-free approach is preferred.

Building from the above assumption of correlation within the objective function, as well as the objective function being a function of n real values as seen in section 1.1, we can separate the inferable information into two categories. The first is dimensional information, for example gradient information for continuous domains. Dimensional information can be inferred by sampling multiple solutions, changing the value along a single dimension and comparing the value of the objective function between samples. In discrete cases this is less useful as the feasible set for each dimension is not necessarily associated. If an objective function has only got dimensional information, we can find the optima by optimising each dimension individually using a gradient descent algorithm, and the function is then convex. In non-convex problems we also find the second information category, namely inter-dimensional information, or how dimensions effect each other. This kind of information is much harder to infer, but many popular heuristics use this information. For example in genetic algorithms, the mutation operator is used for local exploration of solutions, this is comparable to dimensional information. On the other hand the crossover operator, used to merge two solutions together and create child solutions related to both, can be considered as inter-dimensional information. A naive crossover operator works by swapping a section of a solution's vector with another solution's vector. This operator then creates two different offspring to the original solutions, while maintaining the relative order of each other's solutions as shown in figure 3. Maintaining the same order within the swapped subsets is critical for GA to effectively work, as it enables the child solution to maintain the values of the parent solution which are only effective in combination with other values from correlated dimensions. This importance has been shown in research where the crossover operator is accredited for exponential speedups to a naive mutation only algorithm [36].

3.3 Multi-Objective Capabilities and Branching

As mentioned in section 3.1 a probabilistic solution generation will be employed. Therefore, methods such as crossover used in GA will not be applicable, as solutions will be generated directly from a probability matrix rather than propagated from parents. In order to try harness inter-dimensional information as referred to in section 3.2 a branching mechanism will be introduced.

A branch will represent a set of transitions to be taken together, allowing for correlated values across dimensions to be kept together. This will extend how algorithms such as ACO generate the paths for their ants. In a similar fashion a probability matrix will be calculated for every possible transition, and an algorithm will iterate through every dimension, choosing the next transition to be taken by the agent. Unlike ACO, each possible transition will represent a transition tree, where the value stored in the probability matrix is the sum of the probabilities of each branch of the transition tree. Once a transition is chosen from the probability matrix, a further choice will be made between each available branch for the transition tree. This choice will range from the regular transition or add on up to n additional transitions from pre-defined branches, where n will be a user defined max tree length. Figure 4 shows the transition tree for the transition $A \rightarrow D$ in a default 4-dimensional TSP instance, the transition tree on the right shows that on choosing the transition $A \rightarrow D$, the solution generating algorithm will have to make a further choice of the possible branches: $\{(A \rightarrow D), (A \rightarrow D \rightarrow C), (A \rightarrow B \rightarrow C), (A \rightarrow B \rightarrow C \rightarrow D)\}$. To adhere with problem constrains, the solution generating algorithm will have to ensure considered branches are valid, where a branch will only be taken if the extra transition it invokes is a valid one. In order to add a branch to a transition tree, a heuristic using a selection operator will be invoked on seen solutions. This will sort the solutions and allow either the top ranked ones, or any solution above a threshold to add branches representing their solution. Further practical details of this branching heuristic will be explained in section 4.2.

The aim of using a transition tree to enable branches to be considered in the probability matrix is twofold. Firstly this will enable exploitation of promising solution spaces. As branches will only be added for promising solutions, it will increase the probability that future generations create solutions based on these promising solutions. Furthermore, the inter-dimensional information of the seen promising solution will be maintained by adding a branch representing multiple of its dimensional values. As such this will allow for local exploration around promising solutions hopefully helping converge on local minima. Secondly branching will enable multi-objective optimisation innately. Rather than having to use multiple agents as seen in MARL, or multiple ant colonies as used in multi-objective ACO, the branches added will represent promising areas of the search space. As such it will enable simultaneous exploitation of multiple niches explored by multiple differing branches. The main mechanism of this is because it bypasses side-effects of the Markov property, where in traditional probability matrices used in RL or ACO, solution generation will converge on one optima with opportunity of local exploration around the optima due to the induced stochastic nature of the chosen heuristic.

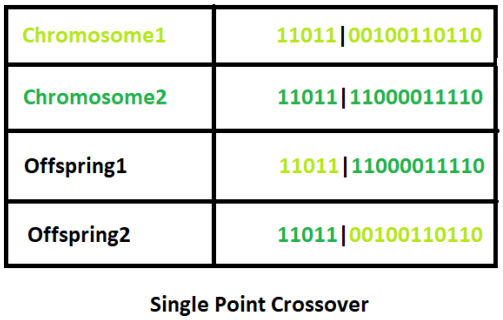


Figure 3: Single point crossover for a binary string taken from [37]

In traditional probability matrices, if two local optima have been explored, the probability matrix will reflect this by having two probable transitions at each dimension, representative of the transitions needed to reach each optima. As a consequence, the probability that a solution is generated which represents either of these two optima is small. This is due the contextual nature of the final solution, where its value may be reliant on the combination of multiple dimensions having specific values. As such when generating a solution from a probability matrix having information on multiple minima, it will generate a solution with a random distribution of good transitions required for both optima, which then loses the inter-dimensional information needed for either optima. As seen in figure 4 a local minima is found at point A, and the global optima is found at point B. When merging these two solutions, either through a crossover technique, or through a probability matrix containing information on both these points, its highly probably point C is the child. Where having merged aspects of both solutions has led to a far worse solution. The use of branching will force generated solutions to come with the context necessary to generate their minima, hopefully negating the generation of poor solutions like solution C.

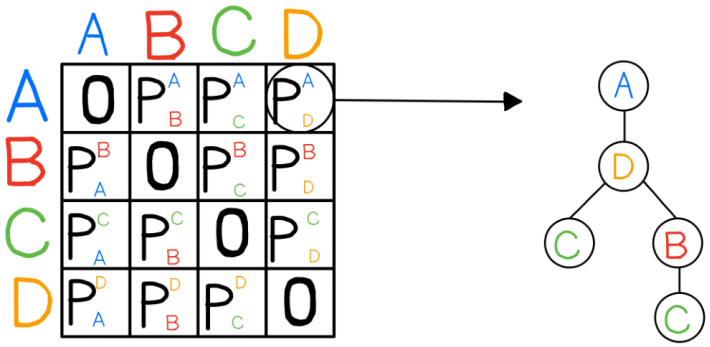


Figure 4: Probability matrix for default 4 city TSP, with transition tree shown for the transition $A \rightarrow D$

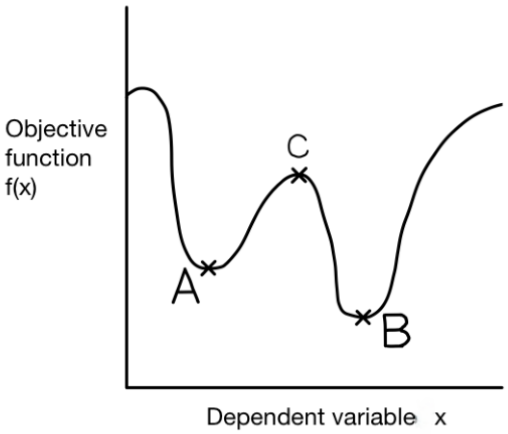


Figure 5: Plotted objective function for a one dimensional problem

3.4 Momentum

When calculating the probability matrix for each transition, two factors will be considered. Namely the best result found by a particular edge, in order to exploit promising regions, as well as a correlated score, similar to the pheromone ants lay down in ACO or Q-values from RL. Both the best historical value and the correlated score for a specific transition will be multiplied together and divided by the sum for all transitions in order to get a resulting probability for that transition. We can formalise this under the form:

$$p_{ij} = \frac{{\tau_{ij}}^{\alpha} + {\eta_{ij}}^{\beta}}{\sum_{j \in \Lambda} {\tau_{ij}}^{\alpha} + {\eta_{ij}}^{\beta}} \tag{2}$$

Where p_{ij} is the probability of taking the transition j from dimension i, for the TSP this could be considered the edge connecting city i to city j. τ_{ij} refers to the correlated score or Q value for the transition j from dimension i, and η_{ij} refers to the best historical score for the transition j from dimension i. α is used in order to change the importance of the correlated score, and similarly β is used to change the importance of the best historical score.

In order to add branch transitions efficiently, added branches should not be their own separate transitions, and rather a list of nodes pointing to existing transitions within the initialised transition matrix. As such, when calculating the probability of the branches, they will inherit the probability of the transitions they point to. This enables scored solutions using branches to update the base transition information of the transitions mapped by the branch, such as their Q-value, their correlation score and their historical best. In order to efficiently use the transition trees, the probability displayed in the probability matrix for a transition represents the sum of all probabilities for each branch in its transition tree. When the solution generating algorithm then selects a transition from the probability matrix, it then makes another choice to select which branch to take, where each node in the transition tree has got its own individual probability, and the sum of the probabilities of its branch and leaf nodes. In order to not inflate the transition tree probabilities, as these will be the sum of an unknown amount of transitions, a momentum characteristic is added when calculating branch probabilities. The momentum value is unique to each transition and branch, and is used to measure how productive a branch or transition is. This will be a basic heuristic, which is slowly reduced every time its branch or transition is used when generating a solution. If the score associated to a solution is better than the previous best score seen for the transitions its made up of, their respective momentum values will get boosted, similarly if a branch is used the momentum of the branch is boosted. As such this encourages exploration down branches which result in good quality solutions, inflating the probability of transitions containing productive branches, but if a branch is unable to produce good quality solutions, the momentum value drops to 0 and the branch is effectively discarded.

Momentum serves as a tool to tackle non-convexity, whereby it encourages exploitation as long as better solutions are found within the minima. But once there is a high chance the minima has been found and no new improved solutions are found, the branch is gradually discarded allowing for exploration in other areas of the solution landscape and reducing redundant computations in an explored area of the landscape. This is a feature inspired by B&B, where no redundant computations are made, a problem often found in MOEAs. Algorithms such as ACO or GA offer little opportunity for further exploration once an optima has been converged upon, leading to many redundant computations within a optima rather than further exploration. In combination with branching, this momentum heuristic will hopefully enable an effective balance between exploration and exploitation, and could be extendable to existing RL or ACO algorithms as an option instead of multi-agent RL or multi-colony ACO.

3.5 Algorithm Overview

Algorithm 1 below shows an overview of the proposed algorithm. It follows a structure similar to that of ACO and RL and so could be extendable to either of these algorithms. In section 4 different operators used to update the transition parameters, calculating the probability matrix and adding branches will be explored. Its important to note that this algorithm has got more overhead to these traditional algorithms as the transition matrix is a matrix of trees, hence efficient matrix computations are harder to implement although still doable with clever use of date structures. Finally, as mentioned in 3.1 this algorithm will be population based, this has the combined benefits of a more well rounded initialisation and lends itself to making this algorithm easily parallelised, where the population generation step can be effectively computed by multiple threads for different solutions.

Algorithm 1: Momentum-driven branching agent optimisation algorithm

```
initialiseTransitionMatrix();
best = inf;
for i = 0 to nIteration do
    calculateProbabilityMatrix();
    pop ← [0,..,PopulationSize];
    score ← [0,..,PopulationSize];
    for n = 0 to PopulationSize do
        pop[n] ← generateSolution();
        score[n] ← evaluateSol(pop[n]);
        updateTransitionParameters(pop[n], score[n]);
    end
    sortedScoreIndex ← indexSort(score);
    if score[sortedScoreIndex[1]] < best then
        best = score[sortedScoreIndex[1]];
    end
    addBranches(sortedScoreIndex, pop);
end
return best;
```

Result: best - The best found solution

This above generalised structure requires 8 inputs, altough some may be deprecated based on design choice. Namely correlationReduction and correlationOffset are not necessary if Q learning is chosen as the correlation score. Similarly, discountRate and learningRate are not necessary if Q learning is not chosen as the correlation score.

- **nDimensions:** Needed to initialise the transition matrix and know what size solutions to generate.
- **populationSize:** The number of solutions to generate per iteration, larger takes more computation time.
- **nIteration:** The number of iterations to compute, larger takes more computation time.
- **maxTreeDepth:** The max length for a branch in each transition tree, cannot exceed nDimensions.
- **alpha:** Used to weight the correlation score, a higher value means the algorithm will focus more on exploration or exploitation depending on the correlation score used.
- **beta:** Used to weight the historical best score, a higher value means the algorithm will focus more on exploitation of seen good solutions.
- **correlationReduction:** Reduce correlation score for each transition by this factor, in order to induce exploration by not allowing certain transition probability to grow exponentially.

- **correlationOffSet:** Induce exploration by giving every transition a minimum amount of correlation score, this enables each transition to be chosen despite historical performance in order to explore around the optimal branches.
- **learningRate:** Ratio defining how much to change current Q value by. A higher value means previous values are not considered, whereas a smaller value means new information slightly changes the current value.
- **discountRate:** When calculating a Q-score for a transition t, the best future transition $t+1$ for a transition t is added. This addition is discounted in order to priorities immediate gain. The discountRate represents how important the future transition is relative to the immediate gain.

4 Development

4.1 Initial Correlation Score

To calculate the probability matrix, equation 2 was used. This equation aims to balance modelled local information, through a correlation score and exploitation through the best historical score. Three ways of calculating the correlation score were considered, the first was a basic rank method. Each generation all the solutions were ranked, and a reward was given to the transitions used by the solution relative to its rank. The equation used was $r = 1/\text{rank}$, where r represents the reward, and rank the rank achieved by a solution. The second option was using a Inverse Gaussian centred around the mean of the solution scores, to weight more depending on their distance from the mean. This following equation formalises this:

$$r = a - a \cdot e^{-\left(\frac{x-\bar{x}}{2\cdot\sigma^2}\right)^p} \tag{3}$$

where r represents the reward, a the size of maximum reward, x the solution score, $\bar{x}$ the mean score over the population , σ the standard deviation of the population and p the flatness of the Gaussian. A larger p reduces the value of r for values close to the mean. A smaller value accentuates values close to the mean letting them have a larger impact. When using the rank method or the Gaussian method the reward is directly added to its respective transition. Once per iteration each transition is reduced according to the factor correlationReduction and offset by correlationOffset. This process is shown by:

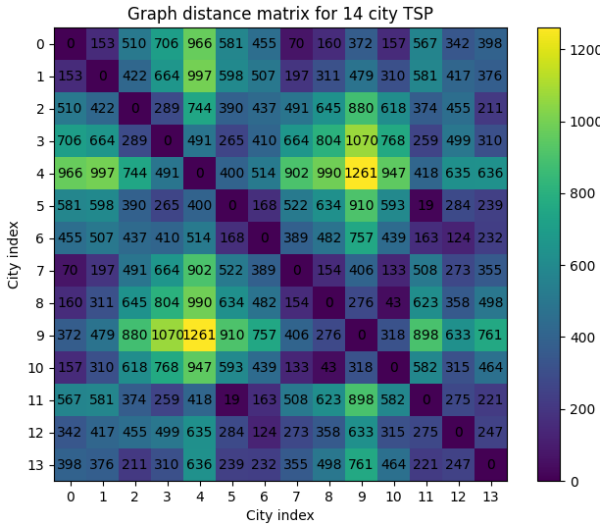
$$C_{ij} = \rho \cdot C_{ij} + \gamma \cdot \min(C_i) \tag{4}$$

where C_{ij} is the correlation for the transition j from dimension i, ρ is the correlationReduction factor, γ is the correlationOffset factor and $\min(C_i)$ is the smallest correlation for all transitions from dimension i. The final option was using Q-values, a method used in RL. This method did not require the Q value to be reduced by equation 4 and would directly modify the correlation score for a transition by the following equation:

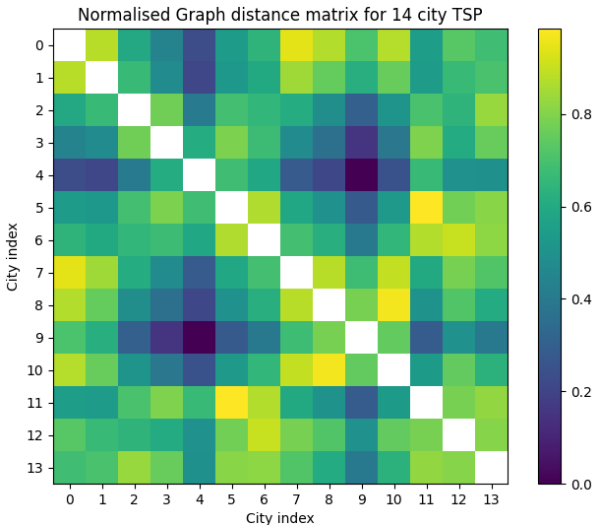
$$C_{ij} = C_{ij} + \rho \cdot (R_{ij} + \gamma \cdot \max(C_j) - C_{ij}) \tag{5}$$

Where C_{ij} is synonymous with the usual notation of Q-values $Q(s,a)$ and has the same definition as in equation 4. ρ is the learning rate, modulating how much new information affects the Q value, and γ is the discount rate, discounting possible future information. Finally $\max(C_j)$ is the transition with the largest Q-value from dimension j, this enables the Q value to have a look ahead property, as its value is dependent on the possible transitions it leads to.

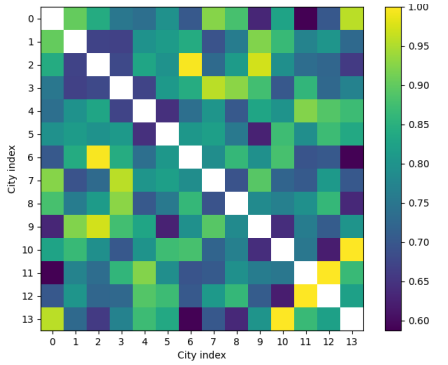
These three methods were tested on a small TSP instance of 14 cities, called burma14. Figure 6 shows experiments to deduce which correlation method to use. Subplot (a) shows the local information for burma14, these values have been normalised in subplot (b), where darker values represent longer transitions, and lighter values represent shorter transitions. Subplots (c), (d) and (e) represent the calculated correlation matrix after generating and scoring 1000 random solutions using the ranking method, an inverse Gaussian and Q-values respectively. As



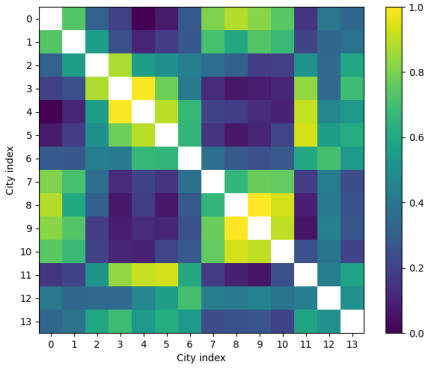
(a) Distance matrix for 14 city TSP instance



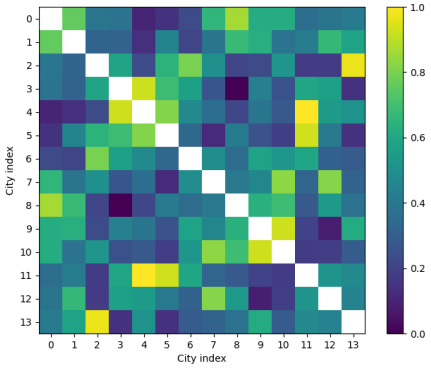
(b) Normalised distance matrix for 14 city TSP instance



(c) Normalised correlation values calculated by ranking solutions



(d) Normalised correlation values calculated using an inverse Gaussian



(e) Normalised correlation values calculated using Q-values

Figure 6: Comparing different methods of calculating correlation score

shown in equation 2, the edge probability is calculated as a factor of both the global best found score for a transition representing global information, and the transition correlation score, ideally representing local information. Looking at subplot (c) we can see the rank method was unable to capture the local trends shown in subplot (b). On the other hand subplot (d) and (e) were able to capture some parts of the local information. Comparing (e) and (d) we can see that the Q-value method was less effective than the inverse Gaussian. Although this may be because of the parameter values, namely a learning rate of 0.3 and discount rate of 0.6 for the Q-value, as such reducing the learning rate could lead to values more representative of the local information. The inverse Gaussian as seen in equation 3 was parameterised with a value for p and a of 1. Furthermore the correlationReduction factor was 0.9 and the correlationOffset was 1.1. It could be worth lowering the value of p as we can see that the inverse Gaussian method captured information with greater contrast than the actual local information. To conclude using either the inverse Gaussian method or Q-learning would be viable, as these are able to best show the local trends in the landscape.

4.2 Adding Branches

Once the base probability calculations were accurately portraying the landscape, transition branches had to be added. The initial model used in section 4.1 was able to run fully on compact matrices. Three matrices were used, one to store the global best value, one to store the calculated correlation score, and the last one represented the

calculated transition probability. As such solutions could be generated using the probability matrix. By iterating through the number of dimensions, the transition representing the value assigned to each dimension could be chosen by looking at the relevant column in the probability matrix. For constrained problems, an extra array is kept to store the valid rows to look from for each dimension. When adding branches This procedure is made more complicated as future dimensions not seen by the loop may have been assigned by a previously chosen branch. To combat this the solution was initialised to a value indicating no transition has been assigned to it, allowing for a check every iteration to ensure the current dimension is currently unassigned. This process is shown in algorithm 2. In addition when selecting a transition a further choice has to be made by the transition tree as to which branch to take. As every branch has an individual probability, a further choice can be made at every branch node, whether to return itself or choose a further branch. This recursive process allows for any branch within the transition tree to be taken according to their probability. This is shown in algorithm 3.

Algorithm 2: Solution Generation

let *blankVal* be a value $x \notin S$, where S is the feasible set for transitions;
let *solution* be an array of length *nDimensions* initialised to *blankVal*;
let *solutionID* be a stack containing the branches and transitions used;
for $i = 0$ *to nDimensions* **do**
  if $solution[i] != blankVal$ **then**
    continue onto next iteration;
  end
  let *current* be the transition chosen according to the relative transition probabilities seen in *probabilityMatrix*$[i]$;
  $path, key$ = chooseBranch($transitionMatrix[i, current, solution]$);
  appendToStack($solutionID, key$);
  for *pair in path* **do**
    $Dimension, transition = pair$;
    $solution[Dimension] = transition$;
  end
end
return *solution* ;
Result: *solution* - The generated solution

Algorithm 3: Choose Branch

solution is an input representing the current seen dimensions;
transition is an input representing the current transitionID;
let *path* be an emtpy stack;
$key = transition$;
let *valid* be the branches of this *transition* assigned to dimensions not seen in *solution*;
let *probArray* be an array containing the probabilities for this transition, as well as each branch in *valid*;
let *cur* be the random branch chosen according to *probArray*;
if *curisnotthisbranch* **then**
  $path, key$ = chooseBranch($cur, solution$]);
end
appendToStack($path, (curDimension, curTransition)$);
return *path*, *key*;
Result: *path* - a list of (dimension, transition) pairs, *key* - an ID or pointer referring to the branch taken

The tree structure lends itself well to Object-oriented programming. In this project the tree structure was im-

plemented using three classes. A base node class was implemented to provide basic utility global to all branches, such as algorithm 3 needed to choose which branch to take, furthermore this also contained the functions necessary to extend the tree among other things. A treeRoot class was then created which extended the initial node class, this class was used for the original transitions and had extra parameters and functions. TreeRoot objects were the only ones store correlation and historical best information in order to calculate transition probabilities. Finally a treeNode class was implemented, these are skeleton objects which extend the original node class as well as having two additional pointers, one to its parent branch, and a second to the treeRoot object the branch is extending. Having branch nodes point to existing treeRoot objects was crucial to allow for high-dimensional optimisation. An experiment was done where transitions would be stored as a list associated with each dimension, where rather than maintaining the original transition matrix, transitions were chosen by choosing valid ones from the transition list associated with a dimension. A preliminary test showed that depending on which heuristic was used to add branches, the transition list would rapidly grow out of hand. Furthermore creating separate branch objects led to information being lost, this was because information was then specific to a branch, rather than the base transitions the branch was made up of. Using a tree object to point to existing root transitions allowed for a lightweight implementation of branching, whilst not losing any information, a crucial optimisation in order to be effective in high-dimensional OPs.

Three heuristics were considered when adding branches, the first was a simple ranked selection, where the best few solutions were converted into branches. The second was a tournament selection. A method which works by selecting k random solutions, and selecting the best out of the selected ones, this is repeated for as many selections as desired. This method allows worse values to propagate, especially if tournament size is small. The third used a calculated cutoff value, where if a solution scored under it, it would be propagated into a branch. Little to no difference was observed using these three methods, although more research could be expended to compare the use of different branching heuristics. Graphs showing each of their performance can be found in appendix A, although its worth noting they were somewhat variable between runs and all performed very similarly overall. The cutoff strategy did fall off in efficacy for larger problems. Depending on the heuristic used, many or no branches would be propagated, leading to unreliable behaviour, this also changed during a run as the values for the mean, standard deviation and best historical score used to calculate the cutoff fluctuated. As a consequence the solution ranking was used. It is worth noting a massive improvement was noticed when implementing branching, the ability to ensure generated solutions maintained inter-dimensional correlation meant a lot more guesses were productive. A comparison can be seen in figure 7, where figure (a) used top ranked solution branching, and figure (b) had no branching. The following values were used in the runs: nIterations = 100, popSize = 100, alpha = 0.8, beta = 0.2, correlationReduction = 0.9, correlationOffset = 1.1. These same values were used in the runs producing the figures found in appendix A.

Its important to note a specific branching algorithm was used to decide which transitions would be merged in a selected solution. When a solution is selected for branching, the vector containing the individual transitions (SolutionID as seen in algorithm 2) was normalised. The normalisation was done by comparing the best historical value seen at every single transition. After which a certain amount of random pairings relative to the number of dimensions was computed on the transitions under a specified threshold. The paired transitions would then be merged, whereby the first of the two transitions would add a branch pointing to the second transition. In this project a threshold of 0.25 for the normalised best historical values was used, it would be interesting to further research the effect of this threshold on how effective the branching algorithm is, and whether a method such as tournament selection would be viable for selecting which transitions to merge.

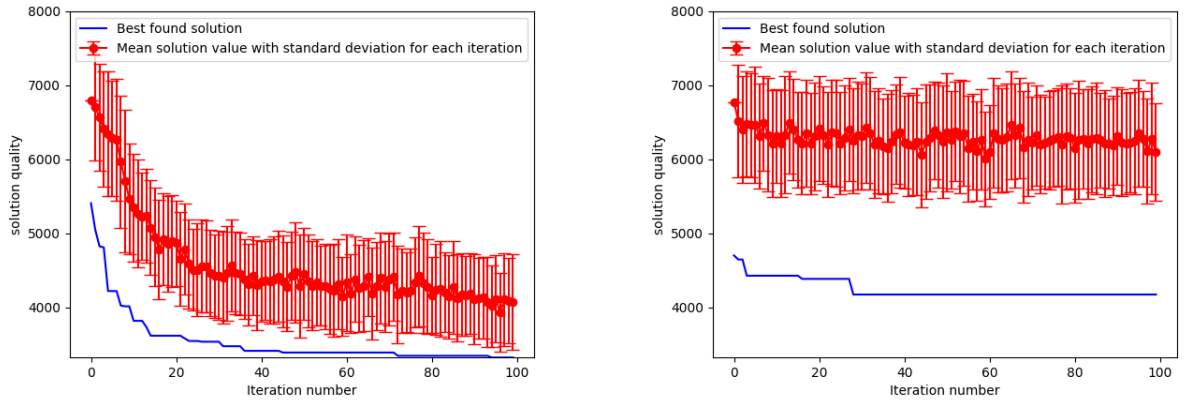
4.3 Adding Momentum

The momentum parameter was added last, with the aim to reduce redundant computations, increase the exploitation of promising solutions, as well as increase exploration once explored regions no longer gave promising results. In order to achieve this the momentum parameter would need to be added to every single branch, and influ-

ence how their probability was calculated. Its important to distinguish that at the moment two types of probability were being calculated. The first was the probability stored by each transition, or treeRoot object, this was done using equation 2 which calculated a value dependent on the specific transition's correlation score and best historical value. The individual probabilities of each treeBranch object was that of the transition it pointed to. The second calculated probability, and also the one used in the probability matrix when generating solutions was the cumulative probability of every single branch within a transition tree. This second calculated probability was calculated by recursively adding the sum off each branch probability to its parent, therefore at any branch, its cumulative probability was not that of the whole tree but rather the cumulative probability of itself and all its child branches. This value is what then allows algorithm 3 to decide which branch to take. The problem with this became that the treeRoot probability would then get inflated by all its branches, with little to no preference for any single branch as their individual probabilities was that of a treeRoot object, or a base transition. The idea behind momentum would be to factor into how an individual treeNode would calculate its probability; where rather than it just copying its transitions probability, it would factor its transitions probability by its personal momentum value in order to get a distinct branch probability.

The momentum value for a transition would be replenished if the transition discovered a solution beating its old best value. Furthermore the probability calculated by equation 2 would be put to the power of the transition's momentum value (as the probability is a float between 0 and 1, a momentum value over 1 decreases the probability, and a value under 1 increases the probability. Therefore the momentum is inverted using the following equation $probability^{maxMomentum-momentum}$ in order to have the desired effect). Every time a transition is used the momentum would be decreased by a set amount, where eventually if enough solutions are tried without getting a fruitful solution, the probability for the branch decreases to 0.

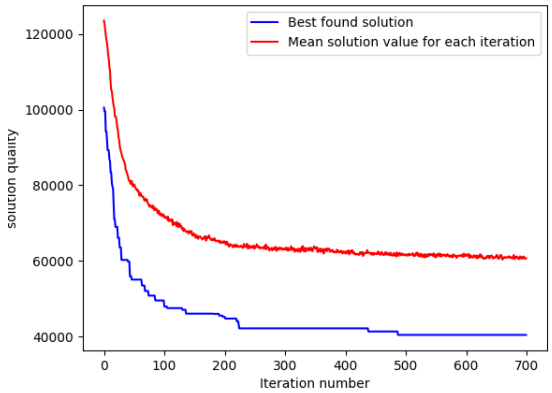
Two implementations were trialled for momentum. The first based momentum on the existing tree structure. To update the momentum for a treeRoot or treeNode object, the score for the found solution would have to be greater than the historical best stored on the treeRoot the transition pointed to. This method proved sensitive to momentum, where a large momentum decrease would rapdly lead to all transitions probabilities converging on 0. I observed that for large branches when a new best solution was found, it would update the best historical value found for every transition pointed to in the branch. This behaviour was desired as it would indicate those transitions were promising, but as a consequence made the momentum parameter a lot harder to update. This was because a branch would not only have to compete with itself, but also the historical best values for all branches having gone through these transitions. To deal with this a second method was implemented, whereby each branch or treeNode



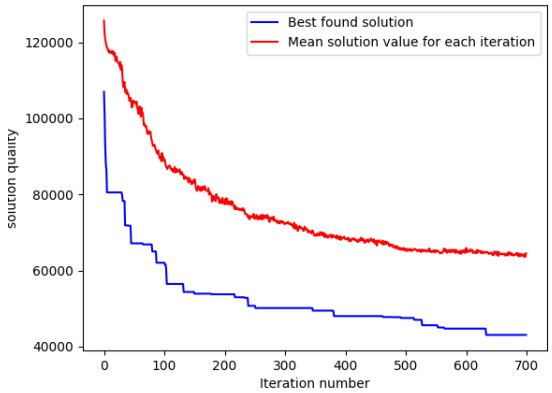
(a) Best found solution, and population mean over 100 iteration, using top ranked solutions

(b) Best found solution, and population mean over 100 iteration, using no branching

Figure 7: Comparing solution convergence with and without branching



(a) Best found solution, and population mean over 700 iteration for 58 city TSP (Brazil58) without using momentum



(b) Best found solution, and population mean over 700 iteration for 58 city TSP (Brazil58) with momentum

Figure 8: Comparing solution convergence with and without branching

object would contain its own historical best value. This value would be inherited from its parent when the branch was created in order to maintain exploitation pressure relative to the niche quality. Therefore, as long as a branch was discovering better solutions relative to its own history it would maintain its momentum to keep exploring. Its important to note the momentum is only a measure of if a branch is producing new better solutions, and that the branch probability is still mostly based on its correlation and historical best as seen in equation 2. This second method was less sensitive to momentum, and I was able to increase the momentum drop off by 4 times while exhibiting similar behaviour. Both these methods were still very sensitive to how momentum was decreased for each branch, as such its important to keep a ratio similar to the problem size, with larger problems having much smaller momentum drop offs.

Figure 8 compares how the proposed algorithm converges on a solution on brazil58, a TSP problem with 58 cities. Figure (a) used the base algorithm using branching, whereas figure (b) used the first method for momentum. Its interesting to note both seemed to find the same quality solution, although the algorithm only using branching had mostly convered, whereas the algorithm using momentum seemed to still be getting better guesses. This suggests that using the correct momentum parameter is crucial to converge correctly, although this would be something requiring further research. The tests done implementing momentum in method 1 and 2 as mentioned above showed that the algorithm behaved very erratically depending on the used parameters. Where too large a momentum drop off would lead to no exploitation as all branches would loose their momentum, and vice versa too little drop off would result in little exploitation as all branches are weighted similarly.

5 Results

The algorithm was compared to GA and RL for TSP instances of 14 cities and 58 cities. The algorithm was tested using both branching and momentum and branching. GA used a tournament selection with tournament size of 5 for all instances. Mutation rate for GA was also constant throughout all tests with a value of 0.05, where a swap mutation operator was used to swap cities together. Finally an inverse crossover strategy was implemented which preserved the relative position of cities whilst allowing for constrained crossover in TSP problems [38]. RL was implemented with an epsilon-greedy search, the learning rate was set to 0.2 and the discount rate was set to 0.4, furthermore the epsilon value was set to 0.1.

For my proposed algorithm, both the branching only variant and the full variant used a max branch length of 30. Alpha and beta were 0.7 and 0.4 respectively, and used a correationReduction of 0.9 and a correlationOffset of 1.1,

apart from problem instances where population size was 50, where correlation offset was 1.025 and correlationReduction was 0.975. The number of branches added per iteration was fixed between 5% to 10% of the population size, being 5 for population sizes of 50, 20 for population sizes of 200, and 35 for population sizes of 700. Finally in the full algorithm the max momentum value was set to 2, with a momentum offset of 0.005 for problems of 200 and 700 population size, and an offset of 0.025 for problems of 50 population size.

popSize	50	Best value found during trial											
Iter num	50	1	2	3	4	5	6	7	8	9	10	Average	σ
GA		3665	3751	3371	3670	3593	3565	3581	3806	3523	3490	3601.5	127.5
Branching		3792	3455	3705	3550	3836	3703	3785	3782	3907	3837	3735.2	138.5
Full		3604	3446	3601	3323	3381	3694	3515	3660	3476	3404	3510.4	125.4

Table 2: Measuring algorithm consistency when solving small TSP instances with a population size of 50 and 50 iterations

As seen above in table 2 all 3 algorithms performed similarly. My proposed solution performed slightly better, having found the best answer of 3323 once in the 10 trials. Interestingly the branching only variant performed worse than GA suggesting that momentum was useful in exploiting good found solutions. Full convergence was unable to occur with 50 iterations, and thats reflected by the varying best scores found in each run. In table 3 below, we can see a larger difference in solution quality found, with my proposed algorithm finding the best solution every single time. The branching only variant was unable to consistently find the optima and would've probably taken a long time to converge on it. As seen in figures 7 and 8 the branching only variant converged very quickly on an optima, but struggled to escape it for a while, as such the values found for the branching only variant are probably the best optima it would be able to find in that particular run. This highlights the effectiveness of momentum to escape minima, whereby adding it led to the reliable finding of the global optima.

popSize	200	Best value found during trial											
Iter num	200	1	2	3	4	5	6	7	8	9	10	Average	σ
GA		3662	3496	3660	4084	3438	3731	3976	3426	3892	3893	3725.8	231.0
Branching		3448	3323	3336	3346	3346	3323	3323	3517	3336	3323	3362.1	66.1
Full		3323	3323	3323	3323	3323	3323	3323	3323	3323	3323	3323	0.0

Table 3: Measuring algorithm consistency when solving small TSP instances with a population size of 200 and 200 iterations

	Burma 14			Brazil 58		
	Best found	Popsize	Iteration num	Best found	Popsize	Iteration num
RL	4410	50	50	76875	200	200
	3968	200	200	64535	500	700
GA	3690	50	50	46749	200	200
	3496	200	200	33609	500	700
Branching	3653	50	50	52905	200	200
	3323	200	200	46959	500	700
Branching and Momentum	3591	50	50	58873	200	200
	3323	200	200	43753	500	700

Table 4: Analysing algorithm performance on difference TSP instances with differing population sizes and iteration count

Comparing the perfomance of the 4 algorithms on larger TSP instances shows that my suggested algorithm and RL converge a lot quicker than GA, and as a consequence of the larger selection pressure, were unable to find as good optima as GA. The full variant for my algorithm performed better than the branching only variant when running a population size of 500 over 700 iterations, suggesting that momentum was able to help exploration efforts escape local minima in the long term. Albeit the branching only algorithm was able to converge on a minima much

faster finding a better solution than the full variant when running 200 iterations with a population of 200. Further testing and fine-tuning of the parameters should be implemented in order to see if the proposed solution can find better optima with less selection pressure. RL performed the worse and undoubtebly came across a local minima it was unable to escape. The difference between the RL algorithm and my suggested algorithm demonstrate the potential effectiveness for branching and momentum in algorithms such as RL and ACO. As seen in figure 8 the proposed algorithm converges very quickly on an optima, displaying similar behaviour to RL or a greedy local search, suggesting that its quick convergence may be related to the high selection pressure applied by the way transition probabilites are calculated.

6 Project Critical Evaluation and Discussion

6.1 Critical Evaluation

The results from the testing in section 5 show that the proposed solution was unable to perform better than existing optimisation algorithms. Despite this, multiple goals defined in section 2.1 were attained. The proposed framework was able to solve Non-convex discrete black-box OPs in polynomial time, where results seen in table 4 demonstrate that adding branching and momentum to a similar framework as RL enabled more effective exploration and exploitation for all problem instances. Furthermore the consistent increase in solution quality as seen in figure 8 (b) demonstrate the global optimisation qualities of my proposed algorithm, most likely due to the momentum heuristic. Furthermore the use of a branching heuristic to leverage inter-dimensional information found in discrete black-box optimisation landscapes lend itself well to a generalised algorithm, as this information is available in all problems of this type. As such the proposed framework seems generalisable for all non-convex discrete black-box OPs. In order to be fully generalisable it would be important to further measure the impact of all the parameters used in the proposed algorithm. Tests showed that the convergence characteristics were very jumpy depending on how momentum was implemented, furthermore the use of an inverse Gaussian to calculate the correlation score was not fully able to capture the local information.

A technical aspect of this project which was not met was the algorithm's multi-objective capabilities. Albeit the current framework using branching and momentum seems extendable to this, and in future work, adding methods such as non-dominated sorting when choosing which branches to add could enable this. The performance of the algorithm when using momentum and branching seemed to easily invoke exploration of diverse optima in the solution landscape, as such I believe these heuristics would be strongly suited for multi-objective optimisation. Although it would be more important to fine-tune the momentum mechanism among other parameters in order to get more consistent performance.

Despite this, a key requirement was for the algorithm to show consistent performance as motivated by the lack of consistent MOEAs for discrete multi-objective black-box OPs [9]. In this regard the algorithm performed very well, with preliminary tests shown in table 3 showed that the final algorithm variant was able to consistently find the optima. This is possibly due to the small problem instance and as such more thorough testing on larger problem instances should be done. Although it would seem this property is inherent to the momentum mechanic, where the full algorithm using momentum seemed to have greater global optimisation abilities than the naive algorithm only using branching. Finally its worth mentioning that the heuristics for branching and momentum are extendable for algorithms including ACO as well as an alternative to multi-agent RL. The combination of these approaches could lead to interesting results and perhaps better exploitation and exploration qualities of these algorithms. As such this also meets a primary requirement for extendability.

6.2 Discussion

A lot of improvement could be made to the proposed framework, but one which stands out is the addition of mutation. During the development process, the strict nature of the branches seemed to limit local exploration. I ob-

served in multiple instances that the algorithm would converge on an optima for hundreds of iterations, yet a further optima could be found by mutating a single dimension. Namely when solving the burma14 TSP instance, the algorithm would often converge on the solution 3336 for a large period before finding 3323, where the only difference between these two solutions was the swapping of a single dimension. Mutation could be tied into the momentum factor, whereby a branch no longer delivering promising solutions could result in having a higher mutation chance triggered by a dropping momentum parameter. This could solve the static nature of the branches, allowing for local exploration in branches, rather than exclusively around them. Adding mutation may be harder to implement than in an algorithm like GA, due to the complex tree structure overlaying each transition. A possible remedy would be to add a new branch to a transition tree if an existing branch is mutated. Thereby allowing for the pointer system to work when updating the transitions the branch points to. Otherwise mutated branches could be added when adding branches directly, where an added branch would have a slight chance to mutate into a different branch adding both the original and the mutated branch to the tree. In either way, the momentum mechanic would be able to quickly pick up on whether the mutated branch had any potential, and further exploit promising mutations.

It's also worth discussing if bounding as seen in B&B would be viable in discrete black-box optimisation. If enough information about a certain transition is found, leading to a high confidence that that transition is not used in any optima, it could be removed. This would allow for search space reduction in a discrete black-box environments. Methods such as attention from deep learning could be leveraged in order to measure a transition's potential along a wide combination of solutions, enabling semi-confident bounding to occur. This could lead to an interesting avenue in research, namely deep reinforcement learning using branching. Its worth mentioning that if better performance can only be achieved by using methods such as deep learning which cost a certain overhead, expensive surrogate-models would also be worth considering, as these algorithms are better prepared to model local information in comparison to the inverse Gaussian used in this project.

7 Conclusion

The aim of this project was to develop a general optimisation algorithm, capable of solving non-convex discrete multi-objective black-box optimisation problems, a notorious subset of optimisation problems which are scarce in information and hard to solve. In order to achieve this, a branching heuristic was implemented which effectively leveraged inter-dimensional information found in these problems. Furthermore a momentum parameter was introduced in order to try reduce redundant computations as well as induce global optimisation when branches fall into local optima.

Preliminary results on varying travelling salesman problem instances showed mixed results, with better performance on smaller problem scales, and poorer performance on larger problem scales. Despite this, the branching and momentum heuristic showed great improvement in exploration and exploitation compared to the baseline probabilistic model. Further work looking at how local information is measured, as well as improving the reliability of the momentum heuristic could lead to significant improvements. Unfortunately, due to the unreliable and erratic performance based on the parameters, this limits the potential for the proposed algorithm to be a good general optimiser.

Other related research could look at including mutation to the branches, or bounding in order to reduce the search space. Otherwise extending the branching and momentum heuristic to existing algorithms could be considered as they've shown an increased ability to exploit and explore solutions compared to a naive probabilistic model. The extendibility of this framework for reinforcement learning would also be an interesting avenue of research, in order to find more decisive heuristics than Q-learning.

Bibliography

[1] M. D. Intriligator, "2. The Mathematical Programming Problem," in *Mathematical Optimization and Economic Theory*, Classics in Applied Mathematics, pp. 8–19, Society for Industrial and Applied Mathematics, Jan. 2002.

[2] C. Carissimo and M. Korecki, "Limits of Optimization," *Minds & Machines*, vol. 34, pp. 117–137, Feb. 2024.

[3] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *Commun. ACM*, vol. 63, pp. 54–63, Nov. 2020.

[4] D. Liu and J. Wang, "Upgrading of IOT Big Data Governance Scheme in Microenterprise Governance," *Wireless Communications and Mobile Computing*, vol. 2022, p. e9831331, Feb. 2022. Publisher: Hindawi.

[5] S. K. Kiangala and Z. Wang, "An effective adaptive customization framework for small manufacturing plants using extreme gradient boosting-XGBoost and random forest ensemble learning algorithms in an Industry 4.0 environment," *Machine Learning with Applications*, vol. 4, p. 100024, June 2021.

[6] D. S. Hochba, "Approximation Algorithms for NP-Hard Problems," *SIGACT News*, vol. 28, pp. 40–52, June 1997.

[7] M. GAREY, "A guide to the theory of np-completeness," *Computers and Intractability*, 1979.

[8] N. V. Sahinidis, "Optimization under uncertainty: state-of-the-art and opportunities," *Computers & Chemical Engineering*, vol. 28, pp. 971–983, June 2004.

[9] M. Li, X. Han, and X. Chu, "MOEAs Are Stuck in a Different Area at a Time," in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '23, (New York, NY, USA), pp. 303–311, Association for Computing Machinery, July 2023.

[10] C.-Y. R. Huang, C.-Y. Lai, and K.-T. T. Cheng, "CHAPTER 4 - Fundamentals of algorithms," in *Electronic Design Automation* (L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, eds.), pp. 173–234, Boston: Morgan Kaufmann, Jan. 2009.

[11] T. P. Papalexopoulos, C. Tjandraatmadja, R. Anderson, J. P. Vielma, and D. Belanger, "Constrained Discrete Black-Box Optimization using Mixed-Integer Programming," in *Proceedings of the 39th International Conference on Machine Learning*, pp. 17295–17322, PMLR, June 2022. ISSN: 2640-3498.

[12] S. Wang, D. Li, S.-Y. Huang, X. Deng, A. H. Sifat, C. Jung, R. Williams, and H. Zeng, "Real-Time Systems Optimization with Black-box Constraints and Hybrid Variables," Jan. 2024. arXiv:2401.11620 [cs, eess] version: 1.

[13] S. Boyd, "Convex Functions," *Hyper-Textbook: Optimization Models and Applications*, Feb. 2021.

[14] R. Sala and R. Müller, *Benchmarking for Metaheuristic Black-Box Optimization: Perspectives and Open Challenges*. July 2020.

[15] R. Agarwala, D. L. Applegate, D. Maglott, G. D. Schuler, and A. A. Schäffer, "A Fast and Scalable Radiation Hybrid Map Construction and Integration Strategy," *Genome Research*, vol. 10, p. 350, Mar. 2000. Publisher: Cold Spring Harbor Laboratory Press.

[16] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, vol. 19, pp. 79–102, Feb. 2016.

[17] A. Tramontani, "Enhanced mixed integer programming techniques and routing problems," *4OR-Q J Oper Res*, vol. 9, pp. 325–328, Sept. 2011.

[18] K. K. Vu, C. D'Ambrosio, Y. Hamadi, and L. Liberti, "Surrogate-based methods for black-box optimization," *Int Trans Operational Res*, vol. 24, pp. 393–424, May 2017.

[19] K. Deb and D. E. Goldberg, "An investigation of niche and species formation in genetic function optimization," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, (San Francisco, CA, USA), p. 42–50, Morgan Kaufmann Publishers Inc., 1989.

[20] O. M. Shir, "Niching in Evolutionary Algorithms," in *Handbook of Natural Computing* (G. Rozenberg, T. Bäck, and J. N. Kok, eds.), pp. 1035–1069, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[21] M. Balinskyy and A. Khitun, "Traveling salesman problem solution using magnonic combinatorial device," *Scientific Reports*, vol. 13, July 2023.

[22] M. Dorigo and T. Stützle, "Ant Colony Optimization: Overview and Recent Advances," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), pp. 311–351, Cham: Springer International Publishing, 2019.

[23] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theoretical Computer Science*, vol. 344, pp. 243–278, Nov. 2005.

[24] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017. Publisher: Nature Publishing Group.

[25] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, p. 105400, Oct. 2021.

[26] C. Halbwidl, T. Sobottka, A. Gaal, and W. Sihn, "Deep Reinforcement Learning as an Optimization Method for the Configuration of Adaptable, Cell-Oriented Assembly Systems," *Procedia CIRP*, vol. 104, pp. 1221–1226, Jan. 2021.

[27] D. Elavarasan, D. R. Vincent, V. Sharma, A. Y. Zomaya, and K. Srinivasan, "Forecasting yield by integrating agrarian factors and machine learning models: A survey," *Computers and Electronics in Agriculture*, vol. 155, pp. 257–282, Dec. 2018.

[28] Y. Zhao, Z. Yang, Z. Wang, and J. D. Lee, "Local Optimization Achieves Global Optimality in Multi-Agent Reinforcement Learning," May 2023. arXiv:2305.04819 [cs, stat].

[29] T. Bartz-Beielstein, C. Doerr, D. v. d. Berg, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, W. La Cava, M. Lopez-Ibanez, K. M. Malan, J. H. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, and T. Weise, "Benchmarking in Optimization: Best Practice and Open Issues," Dec. 2020. arXiv:2007.03488 [cs, math, stat].

[30] J. Stork, A. E. Eiben, and T. Bartz-Beielstein, "A new taxonomy of global optimization algorithms," *Nat Comput*, vol. 21, pp. 219–242, June 2022.

[31] T. Bartz-Beielstein and M. Zaefferer, "Model-based methods for continuous and discrete global optimization," *Applied Soft Computing*, vol. 55, pp. 154–167, June 2017.

[32] J. Bossek, C. Doerr, and P. Kerschke, "Initial design strategies and their effects on sequential model-based optimization: an exploratory case study based on BBOB," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, (New York, NY, USA), pp. 778–786, Association for Computing Machinery, June 2020.

[33] A. Strickler, O. Castro, A. Pozo, and R. Santana, "An investigation of the selection strategies impact on MOEDAs: CMA-ES and UMDA," *Applied Soft Computing*, vol. 62, pp. 963–973, Jan. 2018.

[34] E. Balkanski, A. Rubinstein, and Y. Singer, "The Limitations of Optimization from Samples," *Symposium on Theory of Computing*, 2017.

[35] R. Alizadeh, J. K. Allen, and F. Mistree, "Managing computational complexity using surrogate models: a critical review," *Res Eng Design*, vol. 31, pp. 275–298, July 2020.

[36] D.-C. Dang, A. Opris, B. Salehi, and D. Sudholt, "A Proof that Using Crossover Can Guarantee Exponential Speed-Ups in Evolutionary Multi-Objective Optimisation," Jan. 2023. arXiv:2301.13687 [cs].

[37] "Crossover in Genetic Algorithm." https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/, June 2019. [Online; accessed 30-April-2024].

[38] G. Ucoluk, "Genetic algorithm solution of the tsp avoiding special crossover and mutation," *Intelligent Automation & Soft Computing*, vol. 8, pp. 265–272, Jan. 2002.

A Varying Branch Addition Heuristics

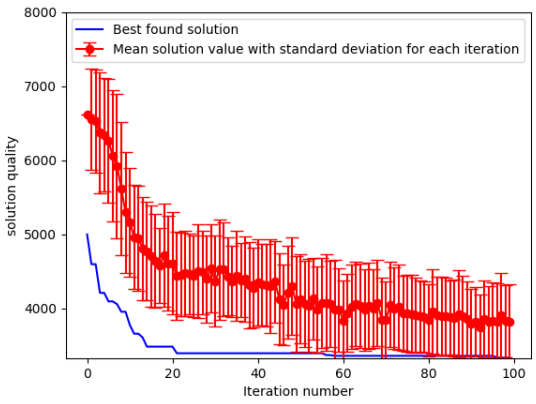


Figure 9: Best found solution, and population mean over 100 iteration, using heuristic branching

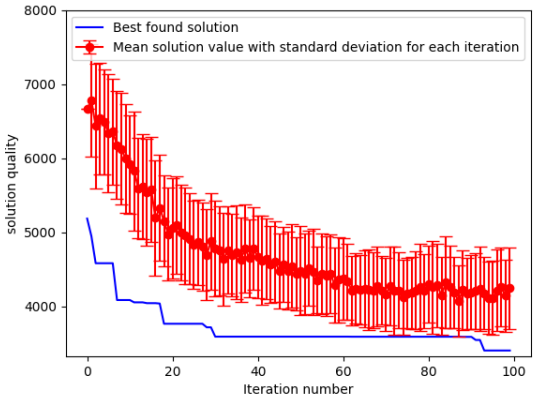


Figure 10: Best found solution, and population mean over 100 iteration, using tournament selection of size 10

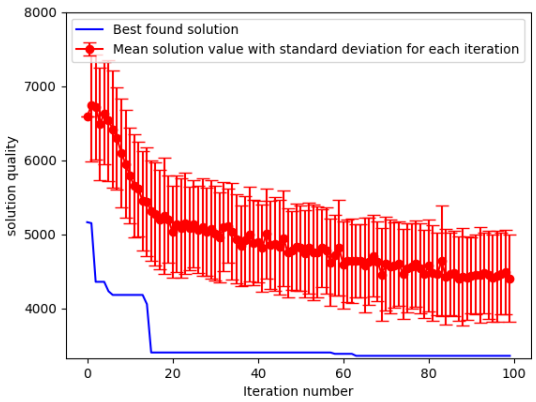


Figure 11: Best found solution, and population mean over 100 iteration, using tournament selection of size 3