Interpretable Supply Chain Optimisation for Inventory Management Problems with Genetic Decision Trees

Jay Geller and Lucia Zhan
Department of Chemical Engineering, Imperial College London, U.K.

Abstract

As machine learning becomes more interlinked with data-driven decision-making processes within critical industries like supply chain management, the transparency and explainability of such models will grow in importance. Decision making guided by black-box optimisation methods has proved to be powerful overall, but high risks are associated with such practices. If stakeholders in charge of decision making are not able to interpret complex machine learning model results, this can negatively impact the confidence that such stakeholders have in acting on the insights given by such models. Decision Trees (DTs) have been known to be highly interpretable in the field. Adding more complexity to DT models can result in boosts in performance, but with the downside of sacrificing interpretability. As a solution, this paper presents the use of Genetic Decision Trees (GDTs, DTs optimised with a genetic algorithm (GA)) as an interpretable machine learning method for solving the Inventory Management (IM) Problem. Different parameters of the GA were investigated, along with different population initialisation techniques and maximum tree depths. The approach was also benchmarked against the most popular IM heuristics. GDTs were shown to outperform all common heuristics whilst maintaining interpretability, and they were also able to pinpoint features of the problem with real-world relevance. Additional, use-case-specific insights were also uncovered, highlighting the promising nature of the proposed approach.

1 Introduction

A supply chain is a network of businesses which work together to procure, manufacture, store and distribute a product or service to fulfill a customer demand (Ganeshan & Harrison, 1995) (Garcia & You, 2015). It comes naturally then that cooperation and information sharing between businesses in the supply chain leads to significantly higher profits overall (Gavirneni, 2005). Over the past few decades, the information sharing revolution has produced exponential amounts of data (Tiwari, et al., 2018). This surge of data opened the door for computational methods such as machine learning (ML) to overtake traditional logistics used for supply chain optimisation, which suffered from their limited ability to forecast demand, and their inability to adapt to real-time data (Makkar, et al., 2020).

Optimising supply chains is crucial for businesses who want to stay competitive. Minimising inefficiencies increases profits by increasing order fulfillment and customer satisfaction. To this end, neural networks (NNs) have been the predominantly used ML method (Toorajipour, et al., 2021). NNs can extract intricate patterns from large datasets due to their inherent complexity. However, this complexity represents a double-edged sword. Whilst NNs and other black-box optimisation methods have demonstrated tremendous success in achieving state-of-the-art performance across various fields and real-world applications, their complexity also limits their interpretability (Zhang, et al., 2021). This trade-off is illustrated in Figure 1 for different ML techniques.

Interpretability of supply chain decisions is crucial for several reasons. Understanding the rationale behind the decisions made by optimisation models increases stakeholders' trust and confidence to act upon the insights given by the model. If inefficiencies arise, interpretability can help with identifying the root causes for issues and hold appropriate parties accountable. Certain industries might also be subject to regulations, in which case interpretability of decisions is critical to avoid non-compliance. Interpretable models have also been shown to be able to reveal important patterns that more complex techniques overlooked (Caruana, et al., 2015).

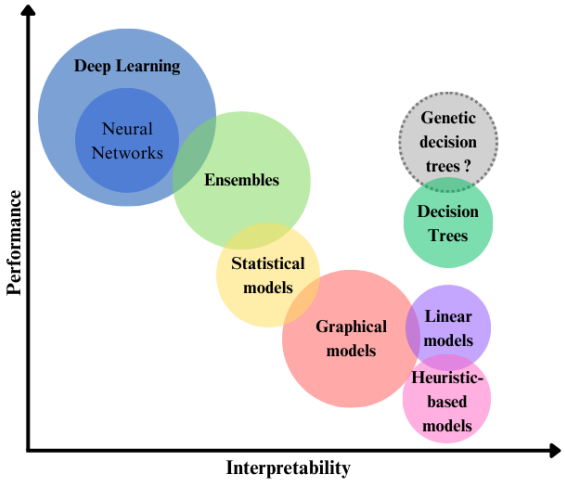

Figure 1: Model performance vs. model interpretability for different machine learning techniques. We expect our GDTs to perform better than individual decision trees and heuristic methods. This study does not investigate how GDTs perform against other methods. This figure was partially adapted from (Yang, et al., 2021).

Within the field of ML, there is a growing body of literature criticising the undervalued importance of interpretability over performance (Baryannis, et al., 2019). Governments across the globe are also developing new legislation around the need for increased transparency in the field of ML. The US with their "*Blueprint for an AI bill of rights*" (The White House, 2022), the EU's "*AI Act"* (European Commission, 2021), and the UK's "*A pro innovation approach to AI regulation*" (Secretary of State for Science, Innovation and Technology, 2023).

The aim of our study is to contribute to this growing, but still underdeveloped body of literature around interpretable methods in machine learning by promoting the use of Decision Trees (DTs) optimised with a genetic algorithm as an interpretable method for supply chain

optimisation, specifically to tackle the Inventory Management Problem.

To our knowledge, whilst genetic algorithms and DTs have been used separately to tackle supply chain optimisation in the past, we are the first to combine their use in this field.

2 Background

2.1 Defining interpretability

Interpretability is widely recognised by the ML community as being one of the most important aspects for improvement in the future of AI, and this sentiment is shared by governments, companies, and the public alike (Carvalho, et al., 2019). Despite this, there is no set consensus for the definition of interpretability. In part because it is difficult to mathematically define interpretability. Different academics and groups will vary in how they characterise interpretable models. As Miller (2019) puts it: "ultimately, it is a human-agent interaction problem". Therefore, in this paper we define interpretability as the ability for a model's result to be predicted by a human; in other words, a human will be able to follow through the model's decision-making process even if they do not understand the rationale behind it.

Blockeel, et al. (2023) provides an extended list of different forms of interpretability which helps understanding how different methods can be interpretable in one or more ways as described below:

(1) Understanding the full model
(2) Understanding aspects of the full model (such as feature importance)
(3) Understanding of a single prediction
(4) Understanding the decision-making process behind a prediction

Within the field of interpretability, the word explainability is sometimes used interchangeably. However, it is important to point out their distinction. Explainable models are interpretable by default, but the opposite is not necessarily true (Gilpin, et al., 2019). We make the argument that explainable models, such as individual decision trees, are those which are interpretable in all four of the above ways.

2.2 Decision trees

Decision trees are sequential models that break down complex decision-making processes into a series of simple tests. Drawing inspiration from nature, decision trees start from a root node where the tree branches out from each sequential test (decision nodes) progressively until reaching a conclusive leaf node. For a visual depiction, see Figure 4.

Early research on decision trees for decision-making was focused on improving their performance, as it was widely acknowledged that individual trees were not able to compete with more complex models such as neural networks (Blockeel, et al., 2023). However, recent studies have shown that certain tree methods such as ensembles and differentiable trees are able to perform as well, if not better, than complex neural networks whilst also being more interpretable (Silva, et al., 2020) (Frosst & Hinton, 2017) (Lundberg, et al., 2020). As a result, it is now widely accepted that such tree methods can compete with neural networks. Even so, these methods have shown that some level of interpretability is traded for their enhanced performance. For example, ensemble tree methods are inherently more complex than an individual decision tree. They lose interpretability when the decisions made by various trees are combined, adding a black-box nature to their decision-making process, and hindering understanding of how different variables contribute to the model's predictions.

It is important to appreciate the trade-off between interpretably and performance of different decision tree methods to understand how further research in the field can advance both aspects of the DT model. In the next section, we propose how genetic decision trees can solve the issue of advanced DT methods inherently losing interpretability, whilst still being able to achieve good predictive performance.

2.3 Genetic decision trees (GDTs)

Pioneered by Holland (1975), genetic algorithms mimic the evolution of living organisms via natural selection and sexual reproduction to solve complex ill-defined problems (Holland, 1992). Through an adaptive process, genetic algorithms take a population of possible solutions (chromosomes), select those which have the higher fitness (those which have proven more useful), and use the genetic operators of crossovers and mutations to evolve said population. This is done iteratively, with the best chromosomes preserved from one generation to the next (elitism).

The combined use of genetic algorithms and decision trees has already been realised by various papers and found to produce robust and scalable predictions (Carvalho & Freitas, 2004) (Bala, et al., 1995) (Fu, et al., 2003) (Vandewiele, et al., 2017). More relevantly, these studies have also found that such an approach can reduce the description complexity of the model, thus improving interpretability of the results.

In this study, these properties of GDTs are leveraged to transform a sequential decision-making problem into a static, data-driven one.

2.4 The Inventory Management Problem

A critical aspect of supply chain optimisation is successfully managing inventory levels to meet uncertain customer demand. The Inventory Management Problem (IMP) describes the challenge of balancing the trade-off between meeting that demand (order fulfillment) and incurring holding costs for excess inventory. To do this, there are three main questions that modelling attempts should aim to answer. First, the frequency in which current inventory levels should be determined. Second, when

should a replenishment order be placed, and lastly how large should this order be.

In order to develop an optimal IM policy for each sequential stage of the supply chain, various constraints must be taken into account. These include supplier constraints (e.g. minimum order quantities, maximum production capacity) and internal constraints (e.g. storage capacity).

The costs associated with the IMP involve replenishment, holding, and shortage costs. The last of which refers to the costs incurred when a short-term shortage of inventory leads to backlog or loss of orders which result in lost profit on sales (Silver, 1981).

3 Methods

A schematic overview of the optimisation approach is presented in Figure 2 as a helpful reference to the reader. The individual steps within the approach will be explained in more detail in the upcoming sections.

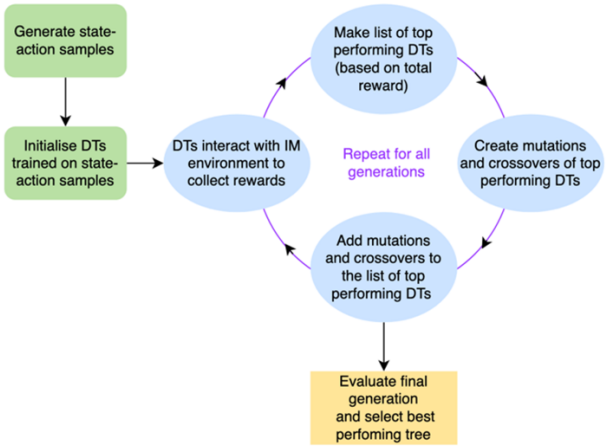

Figure 2: Genetic Decision Tree algorithmic framework.

3.1 Inventory Management environment configuration

To optimise a multi-stage IMP, the inventory management environment provided by Hubbs et.al. (2020) in their Operations Research gym (OR gym) is employed. OR gym is a Python library that is used to develop, test, and compare reinforcement learning techniques.

The IM environment's supply chain structure is illustrated in Figure 3 with the customer at the bottom of the chain. A retailer at stage 0 fulfills customer demand, sourcing products from the supplier at the stage directly above, which may also have another supplier above them. This hierarchy continues until reaching the initial supplier that processes raw materials.

Interacting with this environment involves providing an action (reorder quantity) as input. After that action is taken, the environment evaluates the supply chain and returns the resulting state and a reward. The state contains the resulting inventory levels and actions that were taken to get there, as far back as the value of the maximum lead time. For example, if the maximum lead time is 3 periods, the state will contain the current inventory levels, and actions from the past three time periods. The reward takes into account costs and revenues from all stages, and thus represents the overall profit achieved at a given state. This sequence of taking an action and then observing the results is repeated for the specified number of time periods (i.e., until the end of the episode). The aim of solving the IMP is to maximise the total reward at the end of the episode.

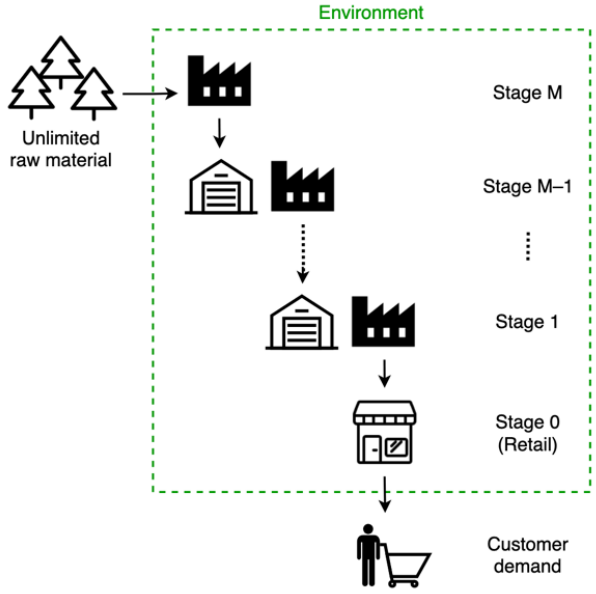

Figure 3: Inventory management supply chain set-up in the OR-gym environment. Partially adapted from (Hubbs, et al., 2020).

For the sake of simplicity, most environment parameters were left in their default state - however, in certain instances that was undesirable. The main reason for this is the tractability of the problem. Given the default parameters, the action space would be comprised of 744,471 actions, which was deemed too computationally demanding a problem, as there's scarce literature available to gauge how powerful the applied method would be *a priori*. Thus, the production capacities and initial inventories were sized down to reduce the size of the action space and state space. As a state is described by prior inventory levels and actions as far back as the biggest lead time, lead times were all deterministically set to 1 to further reduce the size of the state space (for a more detailed description of how a state is described, see (Hubbs, et al., 2020). The rest of the parameters were left in their default state: an episode consisted of 30 episodes, the fluctuating customer demand was drawn from a Poisson distribution, the supply chain under consideration had four stages, and backlogging was allowed. Lastly, replenishment, backlog and holding costs per unit increased from stage 3 to 0, as more refined products cost more to produce and store. For more details on environment parameters, see the Supplementary Information.

3.2 Genetic Decision Tree Regressor (GDTR) class

The most suitable model for this study is a decision tree regressor. We opted to write a custom genetic decision tree regressor (GDTR) class which allows for the use of

crossover and mutation operators. The various methods involved in the GDTR class are described below.

3.2.1 Fit and predict methods

Binary splits were used, that is, the data is partitioned into a left and right dataset based on how the value of the split feature compares to the split threshold. To decide on the best split feature and threshold value, the Sum of Squared Error (SSE) was calculated for each possible split, and the minimum was selected. The formulation for SSE is shown in Eq. 1, where R denotes the part of the dataset going into the right partition, and L denotes that going into the left. Variables $\overline{y_R}$ and $\overline{y_L}$ denote the means of the two partitions.

$$SSE = \sum_{i \in R}(y_i - \overline{y_R})^2 + \sum_{i \in L}(y_i - \overline{y_L})^2 \quad (1)$$

Once the data is split into two sets at a given node, those two parts are split further recursively using the same technique until one of the stopping conditions is met. Two stopping conditions were defined: either the maximum depth needs to be reached, or the dataset needs to be left with only one datapoint (as then there is no point in splitting further). It should be noted that the data was standardised before fitting.

The 'predict' method is essentially a reverse of the 'fit' method, recursively checking how the datapoint to be predicted compares to the threshold value until a leaf node is reached. Then, the value of that leaf node is returned. If at any point the method encounters a criterion which cannot be fulfilled (e.g., $x[2] > 4$ followed by $x[2] < -3$) because a criterion from a node higher up has been modified by a genetic operator, it will default to the right-hand partition.

3.2.2 Crossover creation

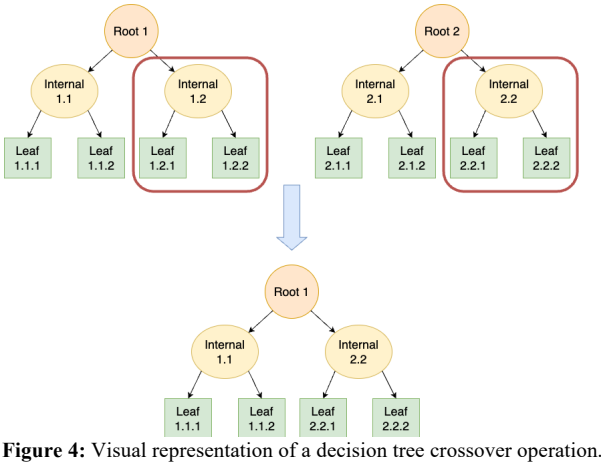

Figure 4: Visual representation of a decision tree crossover operation.

Crossovers are one of the two main genetic operators used to evolve a population. In the context of decision trees, this means exchanging subtrees of two parent trees. For the purposes of this study, the parent trees were selected randomly. Then, one internal node was selected from each parent tree randomly to act as the root node of the subtree to be exchanged. For a visual depiction of the process, see Figure 4. It is to be noted that this operation might lead offspring trees to have a different depth than that of the parent trees, which may be helpful in avoiding suboptimal tree structures due to lack of sufficient exploration.

3.2.3 Mutation creation

Mutations are also genetic operators and, when discussing decision trees, they most often mean modifying one single node's splitting criterion. In this paper the splitting criterion is modified by randomly selecting the split feature, and then perturbing the original threshold by a random percentage between -50% and +50%. The node to be mutated is also selected randomly.

3.3 Genetic algorithm framework

3.3.1 Generating initial training data

Decision trees (DTs) are classed as supervised learning algorithms, meaning that they require a training dataset to be trained on. Such a dataset was generated in three different ways within this study.

(1) State-action pairs are generated randomly, leading to the initial tree population following purely random policies

(2) Half of the state-action pairs are generated randomly, and the other half is generated using the base-stock policy heuristic (for more on this see Section 3.4)

(3) State action pairs are generated fully using the base-stock policy heuristic

For the latter two approaches, each tree initialised using a heuristic was given 200 [state – base-stock action] data pairs to be trained on. It should be noted that stochasticity was also present in the latter two approaches, as the 200 states for which the heuristic was applied were selected randomly.

3.3.2 Fitting the initial DT population

To determine the number of trees in the population, the heuristic outlined in Eq. 2 was followed, where n_s is the number of variables describing a state (6 in this study) and n_a is the number of possible actions (60 in this study). Thus, the initial population was determined to consist of 1320 trees.

$$N_{initial\ trees} = (n_s + n_a) \times 20 \quad (2)$$

Each of these trees were then fit to data generated as described in the previous section, resulting in 1320 different policies.

3.3.3 Genetic algorithm loop

After the initial tree population has been created, a four-step sequence is carried out for 10 generations, after which the final best performing trees are selected:

1. The fitness of each DT's policy is evaluated by having it interact with the IM environment for 10 episodes, reviewing inventory levels at every time period. The end-of-episode rewards are averaged for the 10 episodes, resulting in a final "total average reward" for each of the trees.
2. The top performing DTs are selected to be kept, and the rest are discarded.
3. Crossovers are generated between the trees that were deemed to be the top performers – it is to be noted here that only one of the potential crossovers was generated, and that both parent trees were retained in their original states as well. Next, mutations of the top performing trees were generated and, similarly to crossovers, the original tree was also retained.
4. Newly generated crossovers and mutants were added to the list of top performing trees (so that there were 1320 trees in the population again), and the fitness of each of those trees was evaluated.

As part of the study, multiple setups of this GA were investigated by changing the percentage of trees being retained from the old generation and the percentage of trees which are results of crossovers and mutations. The effects of different maximum initial tree depths were also studied, for all GA strategies.

For a comprehensive tabulated view of all considered setups in this study, see Table 1.

Table 1: Experimental setup; all considered max depth and GA strategy combinations investigated in this study.

Maximum initial depths	Genetic strategy: percentage of trees from the previous generation which are:		
	Retained	Crossovers	Mutations
3, 5, 10, 13	50	30	20
3, 5, 10, 13	40	40	20
3, 5, 10, 13	30	40	30
3, 5, 10, 13	20	50	30
3, 5, 10, 13	10	50	40

Finally, the features of the 5 best performing models were investigated for each initialisation type, max depth, and GA strategy which yields 300 models in total. The investigation included an examination of both the overall occurrences of individual features and the occurrences of each feature when utilised as the initial (root) splitting feature.

3.4 Benchmarks

Heuristic methods are widely used for benchmarking IMPs (Jackson, et al., 2020). In this study we use five of the most popular ones to benchmark our GDTR method. These heuristics are described below:

- *Base-stock policy*: inventory levels are reviewed at each period, and the reorder quantity is placed to bring the inventory levels to a predefined base-stock level
- *(R, S) policy*: same as the base-stock policy, except inventory levels are only reviewed every period R (and the base-stock level is denoted by S)
- *(R, s, S) policy*: inventory levels are reviewed every period R, and if they are found to be below the minimum level s, a reorder is placed to bring the inventory levels to S
- *(r, Q) policy*: inventory levels are continuously reviewed, and if they are found to be below the minimum level r, a reorder of size Q is placed
- *(s, S) policy*: same as the (R, s, S) policy, except inventory levels are reviewed continuously. Also known as the 'Min-max' policy

The heuristics were run in an environment whose configuration was identical to that used for evaluating the GDTR. SciPy's minimise method (Pedregosa, et al., 2011) was used to find the optimal parameters for the heuristic policies, by reformulating them as black-box problems taking the policies' parameters as inputs. Additionally, a random agent taking a random action at each step was also tested.

4 Results

4.1 Benchmarking

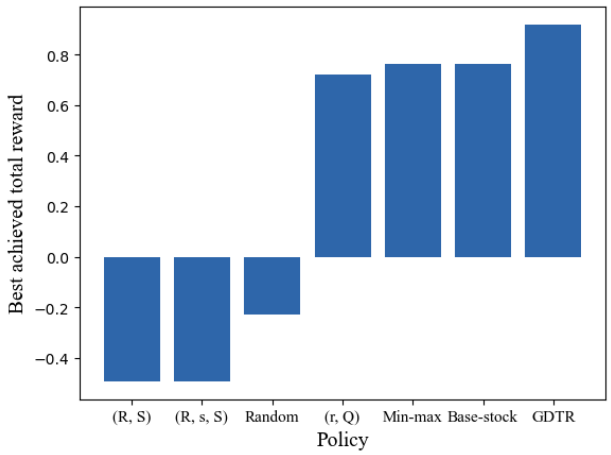

Figure 5: Benchmark test results for 5 different heuristics and a random policy against the Genetic Decision Tree Regressor (GDTR).

Benchmark tests reveal that the GDTR policy performs better than any of the other heuristic approaches outlined in Section 3.4. The highest achieved reward is shown for each policy in Figure 5. Even in the best-case scenario, the random, (R, S), and (R, s, S) policies were not able to break even in costs. In comparison, the (r, Q), min-max and base-stock policies performed much better, yet they were still outperformed by the GDTR. The best GDTR policy outperforms the worst-performing benchmark heuristic, (R, S), by 287%, and it performs better than the base-stock policy by 20%.

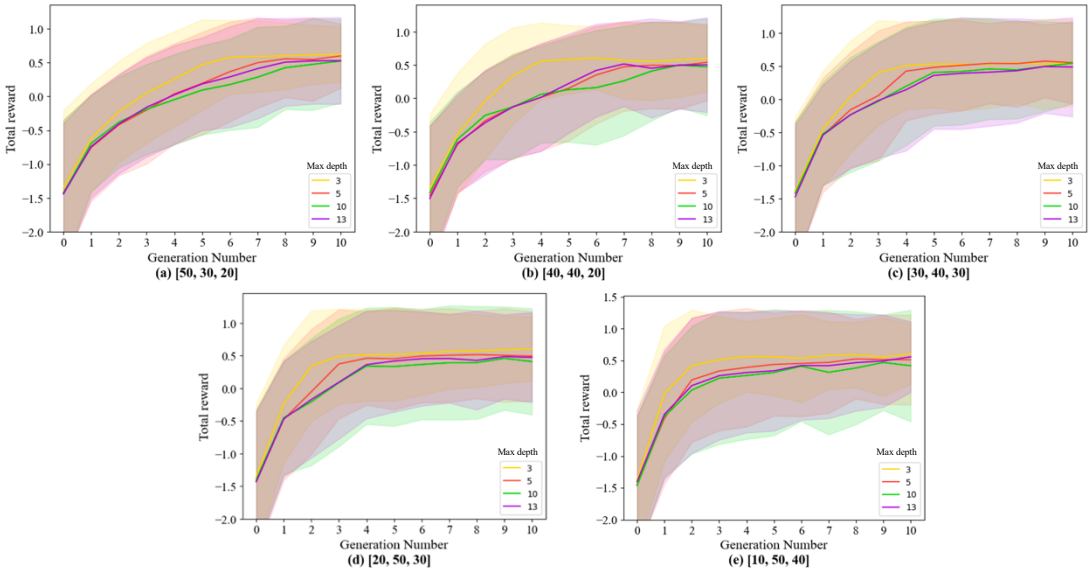

Figure 6: Genetic decision tree regressor reward functions using randomly initialized trees for evolution strategies (a)-(e) and different starting max-depth trees. The different strategies are identified by the number of trees which are [retained, crossover-ed, mutated].

4.2 Investigating the optimal combination of initialisation and GA strategies

Figures 6, 7 and 8 visualise how different combinations of genetic strategies, initial max depths of trees and initialisation methods yield different mean total rewards (solid lines) and standard deviations (shaded regions, +/- 1 σ) across generations. The 'mean of total rewards' refers here to the mean of the rewards achieved by the populations of trees at each generation – it will be referred to as the "reward" in this section for simplicity.

Figure 6 displays how using a random initialisation method affects the performance of subsequent generations of trees for five different genetic strategies and four different max depth of trees. The initial rewards obtained for all strategies and depths were similar in value (-1.5), and notably this initial reward is negative. This is followed by a steep increase in reward from generation 0 and 2, after which most reward functions reach a positive value (breaking even). The steepness of initial gradients is observed to increase with increasing proportion of crossovers and mutations across strategies (a) to (e). Generally, all strategies from generation 8 onwards seem to converge to reward values between 0.0 and 0.5.

For the vast majority of generations, the reward function for max depth 3 performs the best consistently for all strategies and improves at a faster rate than the rest of the depths. From strategy (a) to (e), the shaded regions representing the standard deviation of the rewards steadily increase. Note that no yellow regions are seen at the lower end of the figures, corresponding to trees with initial max depths of 3 performing better than other depths. On the other hand, it seems that for trees with initial max depths of 10, overall improvement is the slowest and with the highest standard deviation (as shown by the green shaded areas).

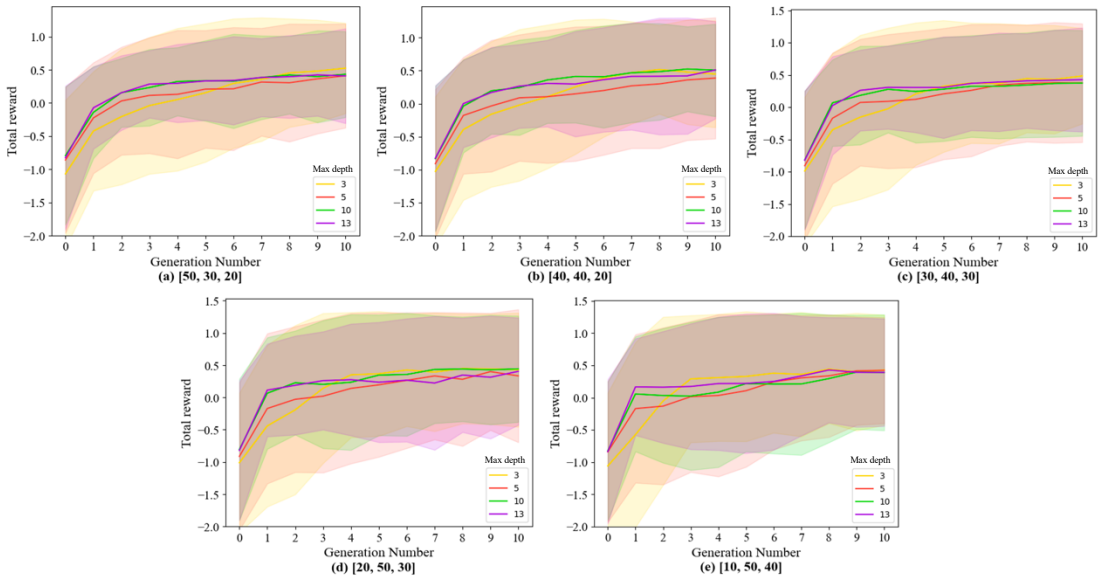

Figure 7: Genetic decision tree regressor reward functions using half random-half heuristic initialized trees for evolution strategies (a)-(e) and different starting max-depth trees. The different strategies are identified by the number of trees which are [retained, crossover-ed, mutated].

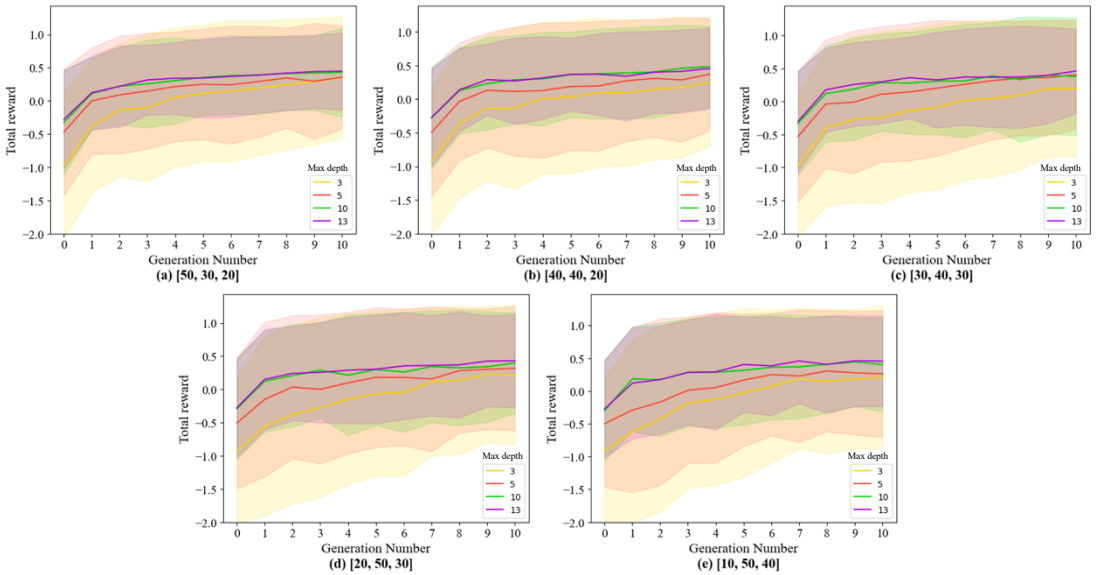

Figure 8: Genetic decision tree regressor reward functions using heuristic initialized trees for evolution strategies (a)-(e) and different starting max-depth trees. The different strategies are identified by the number of trees which are [retained, crossover-ed, mutated].

Figure 7 investigates the same parameters as Figure 6, but for a half-heuristic, half-random initialisation. The general upwards trend across generations is preserved, however, the initial starting point is generally higher (approx. -1, -0.5) than it is for fully random initialisation. Additionally, there is a steep initial gradient of improvement for all GA strategies, not just for the ones where less of the old population is retained.

The trend of lower max depths performing better seems to start reversing, however, the reward functions for the different depths run too closely together to tell exactly. Further, the standard deviations for lower max depth trees (3, 5) have increased when compared to the random initialisation, especially for GA strategies where more of the older generation is retained (strategies (a)-(c)). This may be seen best by observing the increased presence of yellow and red regions in the bottom half of the figures.

Figure 8 shows the results for a fully heuristic initialisation. The general upwards trend of mean rewards across generations for all GA strategies and max depths is preserved with this initialisation type as well. The steeper initial gradient of the reward function is also present, albeit the curve is less steep, as the starting reward is better than that of the previous two initialisation types (approx. -0.25, -0.9). With a fully heuristic initialisation, trees with lower max depths (3, 5) generally perform worse, with max depth 3 visibly performing the worst for strategies (a)-(d). This trend is particularly prominent in earlier generations. Moreover, the standard deviation in achieved rewards increases as max depth decreases, for all GA strategies– the prominence of the yellow and red regions in the bottom regions of the figures is the most noticeable here amongst all initialisation types.

Finally, the different GA strategies seem to be more similar to each other than they are at the previous initialisation types; employing a fully heuristic initialisation decreased the importance of the GA strategy but increased that of the max depth.

4.3 Assessing interpretability of top performing trees

Figure 9's blue bars show the number of occurrences for each of the features used for splits in the top performing trees. The first three features (prior inventory levels) were used much more than the last three (prior actions). There is also a decrease in how much the features were used across the inventory levels, and across actions.

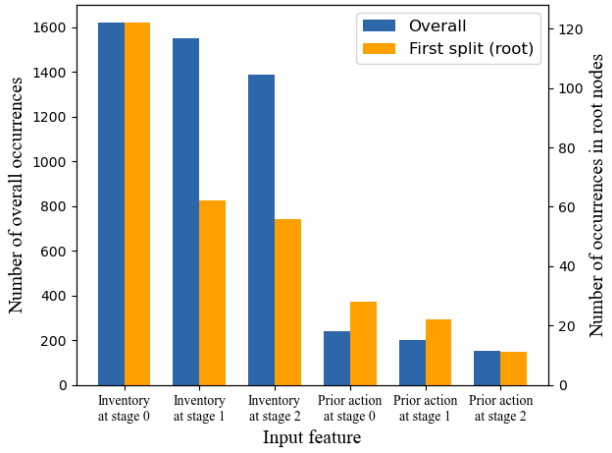

Figure 9: Feature importance for features 0-5 (left to right). Blue: feature occurrence at all root and decision nodes of the top performing trees. Orange: feature occurrence at the root node of the top performing trees. Features represent the splitting criteria at each root or decision node in the decision tree.

The split features at the root nodes were also examined, with the results presented in Figure 9's orange bars. A trend like the one described above was found, that is, the features corresponding to the inventory levels were much more frequently used as a first split than those corresponding to prior actions. However, in this instance the usage of feature 0 was outstandingly high, approximately double the usage of the second most used first split feature (120 vs. 60 occurrences). This trend is

further illustrated in Figure 10, which depicts one of the best performing trees.

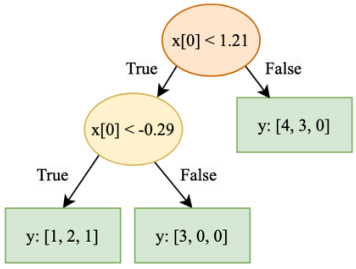

Figure 10: Example output of the approach: an interpretable decision tree of max depth 3. x[i], where i is an integer between 0 and 5, refers to the splitting feature at a specific decision node. For example, x[0] is the standardised current inventory at stage 0 of the supply chain. The variable y refers to the action the tree has outputted at a leaf node, where the values inside the brackets denote the reorder quantity at [stage 2, stage 1, stage 0]. Note that, at stage 3 we have unlimited raw materials, so there is no need for reordering at that stage.

5 Discussion

5.1 Benchmarking

Our genetic decision tree regressor performs better than 6 different heuristics and sequentially improves the performance of an initial population of trees to achieve higher rewards generation after generation. This is in line with existing literature around the ability of genetic algorithms to improve the performance of decision tree models, and it also confirms our expectation that GDTs perform better than heuristic-based models and individual DTs as illustrated in Figure 1.

As mentioned in Section 2.4, the frequency in which inventory levels should be reviewed is an important consideration. For the GDT, this was done at every time period. Regarding the benchmarks, neither of the two worst performing heuristic policies were allowed to do so, suggesting that having regular inventory reviews is advantageous in this setup.

5.2 Investigating the optimal combination of initialisation and GA strategies

Figures 6-8 show that there is a clear trend of improvement for all combinations of strategies as trees evolve from one generation to the next. From this, we can determine that our GDT method was able to improve the performance of an initial population of trees even when such trees were generated using a heuristic, which further supports the validity of our benchmarking results.

The trends seen in Figures 6-8 seem to be of similar shape at a glance. Deeper analysis into the effects of the initialisation method, max depth of trees and GA strategy reveal that they affect the starting rewards, improvement rates, standard deviations, final rewards, and convergence behaviour.

5.2.1 Effect of initialisation method and max depth of trees

The initialisation strategy had a profound influence on the performance of the GDT regressor, and it likely affects how different depth trees will evolve from earlier to later generations. We see this in Figures 6-8, with the same GA strategies producing similar, but different trends for different initialisation methods.

A fully heuristic approach gives reward functions which increase approximately parallel to each other and seem to strongly suggest that the best depth trees to use are of max depth 10 or 13. On the other hand, initialisation methods with a fully random or half random approach tend to show the reward functions intersecting liberally. Although there is still some consistency as to which functions perform better overall, for different strategies there is no clear "winner".

Certain depth trees might perform better than others as shorter trees might suffer from lacking the complexity to fit data accurately (low max-depth, heuristic initialisation), whilst higher depth trees are able to fit to more complex data but run the risk of overfitting and thus performing worse (high max-depth, random initialisation). The higher rates of intersection between different depths' reward functions for the half-heuristic, half-random initialisation seem to illustrate this trade-off.

Using a fully heuristic initialisation also gives higher initial rewards, as expected. However, the reward functions do converge towards mean rewards which are interestingly slightly lower than those achieved by the half-random and fully random approaches. This might suggest that the fully heuristic method is not encouraging enough exploration, leading most of the trees following the algorithm to converge to locally optimal solutions. This, however, does not mean that the fully heuristic initialisation method did not produce top performing individual trees.

Further, the standard deviation of shallower trees increases and their performance decreases as more trees follow heuristically initialised policies. This is potentially due to deeper trees being able to capitalise better on the additional insights provided by the heuristics. This trend is especially prominent in earlier generations, as there was less opportunity to select fitter trees and close the gap between the performance of shallow and deep trees.

5.2.2 Effect of GA strategy

As the percentage of retained trees decreases from strategies (a) to (e), higher initial improvement rates are witnessed when there is randomness in the initialisation, yet the average rewards achieved across strategies at generation 10 are similar. This suggests that whilst increasing the rates of crossovers and mutations might not directly increase the rewards achieved at later generations, it does benefit the evolution of the initial population of trees by increasing the amount of initial exploration the algorithm is allowed. Thus, it can be said that under circumstances where there are limitations on budget, time or computing power, utilising higher rates of genetic operators can reduce the resources needed to reach a converging result. However, this also leads to an increase in the standard deviation of results, which might be undesirable. On the other hand, for the fully heuristic initialisation the effect of the GA strategy is less

pronounced as starting rewards are already higher, as discussed above.

For GA strategies with lower retention percentages, shallower trees have a higher standard deviation when there is some heuristic element present in the initialisation. This points to shorter trees being potentially more sensitive to crossovers and mutations giving them a more profound change than deeper trees would experience. To visualise this, one can think of a tree of depth 3 and another of depth 13 both undergoing a mutation at one of the decision nodes adjacent to one of the leaf nodes. In the grand scheme of things, the tree with depth 3 would be affected more than the tree with depth 13.

5.3 Assessing the interpretability of top performing trees

Features corresponding to inventory levels were found to be better predictors for optimising reorder quantities than features corresponding to prior actions. This was true even with non-zero lead times. Given a specified (albeit stochastic) customer distribution, this would make intuitive sense, and shows the model's ability to uncover real-world insights even when the environment setup presents a rather specific case-study.

The decreasing importance of features as one moves further away from the stage closest to the customer may be explained by their influence on how much of the end customer's order is getting fulfilled. The more an order is fulfilled, the less money is lost to backlogging. What makes this especially important at stage 0 is that replenishment, backlogging and holding costs are higher the closer one is to the end customer (more refined products cost more to store and produce), which means even smaller mismanagements lead to higher losses. Additionally, specifying suboptimal reorder quantities closest to the end customer could create a ripple effect, not only affecting that stage, but constraining the maximum achievable profit of all preceding stages. The exceptionally high occurrence of feature 0 (the inventory level at the stage closest to the customer) as the first, most influential split feature of trees further emphasises this point.

By visualising our best performing trees (Figure 10), we can clearly follow through the decision-making process behind individual predictions. This being that whenever the inventory levels at stage 0 are above a certain threshold, the model predicts that there is no need to reorder at that stage. It does, however, predict the need for reordering at higher stages, probably in anticipation of future demand.

Since the model's decision-making process can be easily navigated by a human, the method was not only proven to yield good rewards but was also found to be highly interpretable.

5.4 Limitations and further research

One of the most relevant limitations of this work is the reliance on a theoretical framework (OR-gym) for inventory management. After all, for this method to be leveraged in real-world applications, additional research needs to be conducted to substantiate the practical utility of genetic decision trees. This is understandably a challenge as the real world also introduces much more uncertainty, but it is necessary to expand the confines of this study for the method to be more widely applicable to fields outside of supply chain optimisation.

Time and computational constraints limited the number of generations, initialisation methods, GA strategies and max depth trees that were possible to investigate. As a proof-of-concept study, this work was able to show that genetic decision trees indeed show promise in optimising IMPs, but further studies should expand on the parameters investigated and look at exploring higher generations and tree depths, as well as more test cases for different GA strategies. This is important in order to gain higher levels of confidence that an optimum has been reached, because in our study we had reward functions which ran fairly close to each other. Consequently, it was difficult to guarantee that an optimum had been found given our limited parameter space. Thus, we further encourage the use of new parameter investigations, for example, introducing different genetic operators.

A further study on benchmarking GDTs against more advanced methods would be crucial to gauge the gap between the proposed method and those at the forefront of the ML field. This includes benchmarking studies against black-box optimisation methods (namely deep neural networks), and other popular methods such as ensembles. This would aid stakeholders in making well-informed decisions when choosing the appropriate optimisation method for their specific application based on the trade-off they're willing to make between interpretability and performance.

Open sourcing our GDTR class can also invite additional collaboration to increase the functionality of our approach. For example, through the creation of more sophisticated interpretation tools to gain further insights into features, splits and decision patterns.

6 Conclusion

This proof-of-concept study investigated a novel approach to optimising an Inventory Management Problem using genetic decision trees.

Benchmark tests against the most relevant heuristics for an IMP revealed that there is promise in using genetic algorithms for improving the performance of tree models whilst maintaining interpretable results.

Parameter tests varying genetic strategies, max depth of trees and initialisation methods were not able to produce clear answers as to which combination of strategies would produce the best performing trees. The stochasticity of the problem also contributes to this problem. So, we have outlined further investigations to be undertaken that could improve the knowledge around which parameters would produce the best predictions.

Nevertheless, final analysis of the top performing trees at the end of the genetic algorithm loop shows that

this approach is indeed interpretable and has the potential to highlight features with real-world relevance. Referring to Section 2.1, the model is interpretable in all four ways described by Blockeel et al. (2023), so there is reasonable ground to claim that this novel approach is not only interpretable, but also inherently explainable. These results affirm the idea that a GDT approach is a promising candidate in the intersection of interpretable machine learning and inventory management optimisation.

7 Acknowledgements

We would like to thank Ilya Sandoval and Niki Kotecha for their invaluable feedback and support.

8 Supplementary information

Access the code for this paper through the following link: https://github.com/jb-gell/Interpretable-SC-Opt-GDT. An extended catalogue of tabulated data of results is also available in the supplementary information document.

9 References

Ganeshan, R. & Harrison, T. P., 1995. *An Introduction to Supply Chain Management,* Department of Management Sciences and Information Systems, 303 Beam Business Building, Penn State University, University Park, Pennsylvania: s.n.

Garcia, D. J. & You, F., 2015. *Supply chain design and optimization: Challenges and opportunities,* s.l.: Computers and Chemical Engineering.

Gavirneni, S., 2005. *Information Centric Optimization of Inventories in Capacitated Supply Chains: Three Illustrative Examples,* Boston, MA: Springer.

Tiwari, S., Wee, H. M. & Daryanto, Y., 2018. Big data analytics in supply chain management between 2010 and 2016: Insights to industries. *Computer and Industrial Engineering,* Volume 115, pp. 319-330.

Makkar, S., Devi, G. & Solanki, V., 2020. Applications of Machine Learning Techniques in Supply Chain Optimization. *ICICCT 2019 – System Reliability, Quality Control, Safety, Maintenance and Management.*

Toorajipour, R. et al., 2021. Artificial intelligence in supply chain management: A systematic literature review. *Journal of Business Research,* Volume 122.

Zhang, Y., Tino, P., Leonardis, A. & Tang, K., 2021. A Survey on Neural Network Interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence,* 5(5), pp. 726-742.

Caruana, R. et al., 2015. *Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-day Readmission.* s.l., s.n.

Yang, G., Ye, Q. & Xia, J., 2021. Unbox the black-box for the medical explainable AI via multi-modal and multi-centre data fusion: A mini-review, two showcases and beyond. *Computer Science and Engineering.*

Baryannis, G., Samir, D. & Antoniou, G., 2019. Predicting supply chain risks using machine learning: The trade-off between performance and interpretability. *Future Generation Computer Systems ,* Volume 101.

T. W. H., 2022. *Blueprint for an AI bill of rights.* s.l.:https://www.whitehouse.gov/wp-content/uploads/2022/10/Blueprint-for-an-AI-Bill-of-Rights.pdf.

E. C., 2021. *The AI Act.* s.l.:s.n.

S. o. S. f. S. I. a. T., 2023. *A pro-innovation approach to AI regulation.* s.l.:s.n.

Carvalho, D. V., Pereira, E. M. & Cardoso, J. S., 2019. Machine Learning Interpretability: A Survey on Methods and Metrics. *Electronics,* 8(832).

Miller, T., 2019. Explanation in artificial intelligence: Insights from the social sciences.. *Artificial Intelligence.*

Blockeel, H. et al., 2023. Decision trees: from efficient prediction to responsible AI. *Frontiers in Artificial Intelligence,* Volume 6.

Gilpin, L. H. et al., 2019. Explaining Explanations: An Overview of Interpretability of Machine Learning.

Silva, A. et al., 2020. Optimization Methods for Interpretable Differentiable Decision Trees in Reinforcement Learning. *Proveedings of the Twenty Third International Conference on Artificial Intelligence and Statistics 2020,* pp. 1855-1865.

Frosst, N. & Hinton, G., 2017. Distilling a Neural Network Into a Soft Decision Tree. *Comprehensibility and Explanation in AI and ML .*

Lundberg, S., Erion, G. & Chen, H., 2020. From Local Explanations to Global Understanding with Explainable AI for Trees. *Nature Machine Intelligence,* 2(1).

Holland, J. H., 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press.

Holland, J. H., 1992. Genetic Algorithms. *Scientific American,* 267(1), pp. 66-73.

Carvalho, D. R. & Freitas, A. A., 2004. A hybrid decision tree/genetic algorithm method for data mining. *Information Sciences,* 163(1-3), pp. 13-35.

Bala, J., Huang, J., Vafaie, H. & DeJong, K., 1995. *Hybrid Learning Using Genetic Algorithms and Decision Trees for Pattern Classification..* s.l., s.n.

Fu, Z. et al., 2003. A Genetic Algorithm-Based Approach for Building Accurate Decision Trees. *INFORMS Journal on Computing,* 15(1), pp. 3-22.

Vandewiele, G. et al., 2017. A Genetic Algorithm for Interpretable Model Extraction from Decision Tree Ensembles. *Trends and Application in Knowledge Discovery and Data Mining.*

Silver, E. A., 1981. Operations Research in Inventory Management: A Review and Critique. *Operations Research,* 29(4), pp. 628-645.

Hubbs, C. D. et al., 2020. OR-Gym: A Reinforcement Learning Library for Operations Research Problems. *Artificial Intelligence.*

Jackson, I., Tolujevs, J. & Kegenbekov, Z., 2020. Review of Inventory Control Models: A Classification Based on Methods of Obtaining Optimal Control Parameters. *Transport and Telecommunication,* 21(3), pp. 191-202.

Pedregosa, F. et al., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research,* Volume 12, pp. 2825-2830.

Supplementary Information

Table SI.1: Environment configuration

Parameter name in environment	Parameter meaning	Parameter value
$periods$	Total time periods in one episode	30
I_0	Initial inventory for the stages	[3, 2, 2]
p	Unit price for final product	2
r	Unit cost for replenishment order at each stage	[1.5, 1.0, 0.75, 0.5]
k	Backlog cost or goodwill loss (per unit) for unfulfilled orders (demand or replenishment orders)	[0.10, 0.075, 0.05, 0.025]
h	Unit holding cost for excess on-hand inventory at each stag	[0.15, 0.10, 0.05]
c	Production capacities for each supplier	[4, 3, 2]
L	Lead times in between stages	[1, 1, 1]
$backlog$	Allow backlogging?	True
$dist$	Specify distribution for customer demand	Poisson
$dist{-}param$	Parameter for distribution (mean)	1.2
$alpha$	Discount factor to account for the time value of money	0.97
$seed{-}int$	Seed for the random state	1

Table SI.2: Best results for each starting depth and GA strategy, for random initialisation

Max depth GA strategy	3	5	10	13
[50, 30, 20]	0.840899	0.85261	0.845541	0.840391
[40, 40, 20]	0.835512	0.881939	0.83718	0.836286
[30, 40, 30]	0.844699	0.83379	0.831822	0.862833
[20, 50, 30]	0.843766	0.839313	0.829334	0.836876
[10, 50, 40]	0.840979	0.885388	0.840979	0.885388

Table SI.3: Best results for each starting depth and GA strategy, for half random, half heuristic initialisation

Max depth GA strategy	3	5	10	13
[50, 30, 20]	0.848432	0.858084	0.884147	0.825404
[40, 40, 20]	0.840227	0.881939	0.852574	0.838547
[30, 40, 30]	0.844699	0.860554	0.889757	0.862833
[20, 50, 30]	0.843766	0.942868	0.84135	0.836876
[10, 50, 40]	0.840979	0.846089	0.840176	0.837421

Table SI.4: Best results for each starting depth and GA strategy, for fully heuristic initialisation

Max depth GA strategy	3	5	10	13
[50, 30, 20]	0.807047	0.83428	0.868323	0.880887
[40, 40, 20]	0.827749	0.867596	0.912032	0.889922
[30, 40, 30]	0.854648	0.887658	0.837331	0.888835
[20, 50, 30]	0.866348	0.903953	0.878274	0.909002
[10, 50, 40]	0.909659	0.920159	0.923882	0.840091