Model-based Genetic Algorithms for Algorithm Configuration

Yuri Malitsky **Horst Samulowitz** **Meinolf Sellmann**
IBM Research, USA
{ymalits,samulowitz,meinolf}@us.ibm.com

Carlos Ansótegui
DIEI
Univ. de Lleida, Spain
carlos@diei.udl.es

Kevin Tierney
DS&OR Lab
University of Paderborn, Germany
tierney@dsor.de

Abstract

Automatic algorithm configurators are important practical tools for improving program performance measures, such as solution time or prediction accuracy. Local search approaches in particular have proven very effective for tuning algorithms. In sequential local search, the use of predictive models has proven beneficial for obtaining good tuning results. We study the use of non-parametric models in the context of population-based algorithm configurators. We introduce a new model designed specifically for the task of predicting high-performance regions in the parameter space. Moreover, we introduce the ideas of genetic engineering of offspring as well as sexual selection of parents. Numerical results show that model-based genetic algorithms significantly improve our ability to effectively configure algorithms automatically.

Many algorithms have parameters that greatly affect their performance. These parameters allow algorithms to adapt to different usage scenarios and provide end users with greater flexibility in customizing software specifically for their needs. However, finding good parameters is difficult and time-consuming because each evaluation of the algorithm being configured is expensive, often taking from seconds or hours to assess a single input (of which there are many).

In comparison, typical local search approaches for standard benchmark problems are able to evaluate tens of thousands of solutions per second. Two types of configurators have emerged: non-model-based and model-based approaches. Non-model-based techniques generally rely on either an experimental design methodology, such as the case in CALIBRA [Adenso-Diaz and Laguna, 2006] and I/F-Race [Birattari *et al.*, 2010], or a meta-heuristic as ParamILS [Hutter *et al.*, 2009] or GGA [Ansotegui *et al.*, 2009]. In contrast, model-based approaches like SMAC [Hutter *et al.*, 2011] use machine learning in order to predict good parameterizations based on samples of the parameter space. We propose GGA++, an automatic algorithm configurator that harnesses the power of model-based approaches within a non-model-based genetic algorithm framework.

Research partially supported by the Ministerio de Economía y Competividad research project TASSAT2: TIN2013-48031-C4-4-P.

1 Gender-based Genetic Algorithm Configuration (GGA)

GGA [Ansotegui *et al.*, 2009] is a genetic algorithm-based approach to general algorithm configuration which has been used in practice for a wide array of applications, such as tuning solvers in SAT [Kadioglu *et al.*, 2010]/Max-SAT [Ansótegui *et al.*, 2014], machine reassignment [Malitsky *et al.*, 2013a] and mixed-integer programming [Kadioglu *et al.*, 2010]). We describe the key components of GGA, but refer readers to [Ansotegui *et al.*, 2009] for further details.

Overview: GGA accepts a parameterized algorithm to be configured (called the *target algorithm*), a set of algorithm inputs, and a performance metric to be optimized. The performance metric can be the execution time of an algorithm or some value computed by the algorithm, for example, the result of a simulation or accuracy of a prediction. GGA then runs its specialized genetic algorithm that executes the target algorithm with different parameters in order to find a good setting. Once GGA has reached the target number of generations, the best parametrization found so far is returned.

Population: A core idea in GGA is the partitioning of the population into a *competitive* and *non-competitive* population. The competitive population is directly evaluated on the target algorithm, whereas the non-competitive population simply acts as a source of diversity. This allows GGA to use strong intensification procedures on one part of the population while remaining diverse enough to avoid local optima.

Selection: A tournament-based selection mechanism is used for evaluating the competitive population. Tournaments are run on disjoint subsets of parameterizations in parallel on a subset of the instances. When tuning runtime, as soon as a set percentage of the parameterizations in a tournament are finished, all other participants in the tournament can be terminated. This saves significant amounts of CPU time that would otherwise be spent evaluating bad parameterizations.

Genome representation: GGA represents parameterizations using an AND-OR tree in order to capture the dependencies between parameters. A node in the tree is a parameter or an AND node that binds multiple parameters together. Several sample trees are shown in Figure 1. In these particular

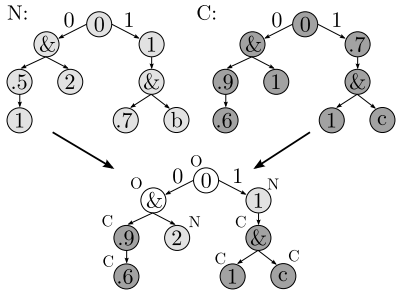

Figure 1: The GGA recombination operator.

trees, the root node is a categorical parameter that activates the left branch when it takes the value 0, and the right branch when it takes the value 1 (think of a parameter that chooses which subroutine is used, with parameters in the left subtree affecting only subroutine '0' and parameters only affecting subroutine '1' in the right tree).

Recombination and mutation: The crossover operator in GGA takes two genomes, one from the competitive and one from the non-competitive population. The competitive genome is a tournament winner, while the other genome is drawn uniformly at random from the non-competitive population. The crossover process is shown in Figure 1, and it can be viewed as a hybrid of the traditional multi-point and uniform crossovers that considers parameter dependencies.

GGA performs mutation simply by selecting random parameter settings and replacing them with values drawn from either a Gaussian (for discrete and continuous parameters) or a uniform (for categorical parameters) distribution.

2 Non-Parametric Models for Predicting High-Performance Parameter Regions

GGA evaluates a parameter setting by running the target algorithm with the parameter setting on a collection of training instances. That is, we conduct a black box optimization with an extremely expensive to run black box. One idea to improve optimization performance is to reduce the high costs of querying the black box directly. To this end, the idea of trying to learn a so-called "surrogate model" that is faster to evaluate and that "approximates" the actual black box has been introduced. The so-called *response surface* methodology [Box and Wilson, 1951], e.g., fits a second degree polynomial to the observed black box output and then optimizes over that polynomial to predict an optimal input for the black box.

A state-of-the-art sequential local search algorithm configurator that uses a surrogate model is SMAC [Hutter *et al.*, 2011]. SMAC directly evaluates algorithm performance for some initial parameter settings and then uses this experience to train a random forest that predicts expected algorithm performance over the parameter inputs. Importantly, SMAC also computes the model's variance/uncertainty in its predictions. As a next point for direct evaluation of the algorithm on the training inputs, SMAC chooses a parametrization for which the model predicts high performance *plus* uncertainty. The model is re-trained using this information, along with several random parameterizations that are also directly evaluated. A new parametrization is then chosen based on the updated model, and this process is repeated until the designated CPU time for configuration is exhausted.

The use of variance in SMAC is very important. First, this leads to an improvement of the model itself by favoring points where the model has little "confidence" in its own predictions. Secondly, a vanilla random forest can never return predictions that exceed the most highly rated prior experience. Consequently, if variance was not used to score new parametrization candidates the system could repeatedly re-evaluate the same parameter settings.

3 Surrogate Models for Genetic Algorithms

The core idea of this paper is to study the use of surrogate models in genetic algorithms for algorithm configuration. In particular,

1. How can surrogate models help find high-quality solutions faster in the context of a population-based local search approach?
2. Are there surrogate models that are more suited for predicting high-performance parameterizations?

There is already work on combining evolutionary and genetic algorithms with model-based approaches (e.g., [Rasheed and Hirsh, 2000; Jin, 2011]). These hybrids employ vanilla machine learning models as surrogates. They consider the various uses of surrogates within genetic algorithms and decide when to re-train the model. In this work we devise specialized surrogate models and tightly integrate them within a genetic algorithm.

The core operation that defines the next fitness evaluations in a genetic algorithm is the crossover, possibly in combination with mutation. Usually, the crossover method is randomized in some fashion. We introduce the idea of using a surrogate model for predicting offspring performance and to "genetically engineer" offspring that are expected to perform better. Note that changing the way new parameterizations are introduced into the population can have a drastic effect on the trajectory of the entire search, as this modifies the delicate balance of exploration vs. exploitation. Within GGA, the question then becomes what percentage of offspring should be selected by the model and what percentage should be produced by the existing crossover operator, and how sufficiently high levels of diversity in the gene pool can be maintained.

Another idea we introduce is that of model-based "sexual selection." As outlined above, GGA maintains two populations, one that competes for the right of mating, and one within which individuals mate with equal probability. Based on the assessment of a surrogate model, winning competitive individuals could choose a mating partner based on some definition of their "attractiveness."

Regarding the second question posed above, based on the developments in algorithm portfolios, where non-parametric, or at least low-bias models, have led to the currently best performing approaches [Xu *et al.*, 2012; Malitsky *et al.*, 2013b], we consider random forest regression to train our surrogate model. However, off-the-shelf machine learning random forests were developed to lead to good predictive performance for *all* inputs to the function to be approximated.

For the purpose of predicting search regions that harbor high-quality solutions, we are much more interested in predicting areas where near-optimal inputs can be found and less so in high-accuracy predictions throughout the search space. To this end, we introduce a new splitting method for random forests that aims at predicting high-quality solutions better than general regression models.

According to the discussion above, the three core algorithmic contributions of this paper are 1. genetically engineering offspring, 2. selection of non-competitive mating partners based on their attractiveness, and 3. building decision trees (that will form random forests) with a bias towards better predictive performance around function extrema.

3.1 Genetic Engineering

The core idea of genetic engineering[1] offspring is very simple. Once two parents have been chosen to produce offspring together, we aim at finding the combination of the parents' genes that results in the fittest offspring as predicted by our surrogate model. Therefore, what happens during genetic engineering is a limited focused search in the restricted parameter space spanned by the two parent genomes. These genomes provide at most two specific values for each parameter to be tuned, or a single value in the case that both parents agree on the value. In general, even such a limited search over a non-parametric model, such as a random forest, is hard:

Theorem 1. *Given natural numbers n, k, maximizing a function $\{\text{true}, \text{false}\}^n \to [0, 1]$ represented as a random forest with k trees is APX-hard, even if the depth of all trees is lower or equal to two and each variable appears in at most three trees.*

Proof. We first reduce Max-SAT to the problem of finding a maximizing input for a random forest. The polynomial construction works straight-forwardly as follows: For each clause in the given SAT formula with k clauses over n Boolean variables, we construct a decision tree. We sequentially branch on the literals in the clause: If the literal evaluates to true, we connect to a leaf node with value 1. If it evaluates to false, we move to the next literal in the clause. If and only if the last literal is also in the clause that evaluates to false, we connect to a leaf with value 0. Each tree therefore returns a value of 1 if and only if the respective clause is satisfied, and 0 otherwise. The random forest, which returns the average value of its constituent trees, thus gives the fraction of clauses that evaluate to true under a given assignment. Since Max-2SAT(3) is APX hard [Ausiello *et al.*, 1999], we get the result. □

Note that this targeted search in the space that two points span in the search space is reminiscent of the "Aufhebung" procedure that produces syntheses in dialectic search [Kadioglu and Sellmann, 2009]. One way to conduct such a search heuristically is path-relinking [Glover *et al.*, 2000], another would be to start a full-fledged local search in the focused search space.[2] Since we are dealing with a very specific representation of the objective function as a random forest, we propose using a targeted sampling method.

For each tree in the forest, we gather all leaves that any potential offspring of the two parents could fall into. This can be done in linear time in the size of the given decision tree. Now, for each leaf we randomly sample a number m of full assignments to the parameters. Note that the leaf only constrains some parameters to be set to the value of one specific parent, the others can be chosen at will. We then evaluate the full assignment using the entire random forest. We repeat this procedure for all reachable leaves in all trees, ultimately returning the assignment that results in the maximum value as returned by the forest.

Theorem 2. *For a given $m \in \mathbb{N}$, the procedure above runs in time $O(s^2 m)$, where s is the size of the given random forest.*

Proof. The procedure requires at most s times m random forest evaluations. Since one evaluation requires linear time as well, the total time is quadratic in the size of the given random forest for any constant m. □

As a final note, when genetically engineering a large percentage of new offspring there is a danger of losing population diversity, as genetic engineering intensifies the search. To alleviate this problem, we remove a slightly larger percentage of the population in each generation and replace these individuals with new, randomly generated individuals. As usual in GGA, these as well as the new offspring are assigned the competitive or non-competitive gender uniformly at random.

3.2 Random Forests for Optimization Forecasting

The previously described algorithm for producing optimized offspring works with any random forest. Off-the-shelf methods for training random forests are designed to approximate the function to be learned well *everywhere*. For our purposes, we do not require an approximation of the underlying function (in our case: target algorithm performance on the training instances given certain parameter inputs). We just need the surrogate model to identify those areas of the parameter space *where we expect high-performance parameterizations.*

When training the random forest, and in particular building an individual decision tree, we consider an alternative splitting procedure that results in higher "resolution" of the tree in areas where performance is high, and a less refined approximation in areas where performance is low. More precisely, we will focus on the top $q = 10$ percent of top-performing examples in each split and try to separate the top performers from the remaining parametrizations as best as possible.

To achieve this we need a valuation function for potential split parameter-value pairs that scores those pairs the highest for which the top performers are best separated from low performers. One way to achieve this is to focus on the partition that would contain more of the top $q\%$ of performers. Within this partition, we want the ratio of high performances over low performances to be maximized. To compute these high

[1] We stress that all of the topics pertaining to genetics here are no more than a convenient analogy.

[2] It is worth noting here that SMAC conducts a local search for finding new candidates that combine good expected performance with uncertainty in the surrogate model.

and low performances, we consider the squared differences of the high and low performing valuations compared to the threshold performance v_h of the qth percentile performance.

Formally, we denote the valued training observations at a current node in the tree under construction as

$$E = \{e_1 = (p_1^1, \ldots, p_n^1, v^1), \ldots, e_r = (p_1^r, \ldots, p_n^r, v^r)\},$$

such that $v^i \geq v^{i+1}$ for all $1 \leq i < r$. Let $h = \lceil \frac{rq}{100} \rceil$, $T = \{e_1, \ldots, e_h\}$, and $U = \{e_{h+1}, \ldots, e_r\}$. For parameter index $1 \leq j \leq n$ and splitting value s we set $L = \{e_i = (p_1^i, \ldots, p_n^i, v^i) | p_j^i \leq s\}$ and $R = E \setminus L$ as the left and right partitions of training examples if we were to split on parameter p_j and splitting value s.

To score the potential split, we now consider the following values: $\mathrm{lls} = \sum_{e_i \in L \cap U} (v^h - v^i)^2$, $\mathrm{rls} = \sum_{e_i \in R \cap U} (v^h - v^i)^2$, $\mathrm{lts} = \sum_{e_i \in L \cap T} (v^i - v^h)^2$, $\mathrm{rts} = \sum_{e_i \in R \cap T} (v^i - v^h)^2$, $\mathrm{ltn} = |L \cap T|$, and $\mathrm{rtn} = |R \cap T|$. We then compute $a_l = \frac{\mathrm{ltn}+\mathrm{lts}}{1+\mathrm{lls}}$ and $a_r = \frac{\mathrm{rtn}+\mathrm{rts}}{1+\mathrm{rls}}$. Finally, we set

$$\mathrm{score}(p_j, s) = \begin{cases} a_l & \mathrm{ltn} > \mathrm{rtn} \\ \min\{a_l, a_r\} & \mathrm{ltn} = \mathrm{rtn} \\ a_r & \mathrm{ltn} < \mathrm{rtn} \end{cases}$$

and split on the parameter-value pair that gives the maximum score. Apart from the splitting procedure, we keep the training of the random forests exactly the same.

3.3 Sexual Selection

The third main algorithmic idea we contribute is the use of a surrogate model like the one developed in the previous section for sexual selection. In biology, the term refers to individuals of a population giving preference to mating with more attractive partners as gleaned from external features that indirectly indicate fitness and health [Wedekind *et al.*, 1995].

In this analogy (and we would like to again stress that it is just that: an analogy), our surrogate model has the same function as the external features: it indicates which mates might be better partners for producing fitter offspring. The procedure we will test in the next section works as follows: After the competitive individuals have been selected in GGA, they choose their partners from the non-competitive subpopulation based on the estimated fitness of the partner.

To assess the latter, we obviously use the surrogate model. If the top competitive individuals are to mate with k partners in a given generation, then they each select k non-competitive individuals in a probabilistic process where a mate is chosen with higher probability the higher the non-competitive's performance as estimated by the surrogate model.

4 Experimental Results

When tackling a problem like automatic algorithm configuration, where achieving any kind of performance guarantee is in fact not just hard but undecidable, experimentation is key. That is, to find out whether a method works in practice, we put it to the test. In this section we first conduct a series of experiments on small black box optimization problems to gain an initial understanding of whether genetic engineering is at all helpful within the context of genetic algorithms, and what percentage of offspring ought to be engineered to achieve the best overall results. We then quickly move on to real-world algorithm configuration problems to determine the impact of adding sexual selection, as well as to evaluate the impact that different surrogate model optimization methods have on the configuration performance. We conclude the experiments with a comparison of the model-based GGA with the unaltered configurators GGA and SMAC.

4.1 Potential of Genetic Engineering and Diversification

As we have seen before, finding the best (as predicted by a random forest surrogate model) offspring for two parents is hard. Moreover, even if we could engineer offspring to perfection, the question arises whether optimizing new individuals in a genetic algorithm is beneficial at all. In this section we try to answer these questions by focusing on low-dimensional global optimization problems which allow us to perfectly engineer offspring.

The use of genetic engineering within GGA poses a particular challenge in terms of balancing the search intensification and diversification. It is possible that the engineering prematurely pushes the search into a local optimum. We therefore devise the following experiment to shed light on the influence of being greedy or relying on randomiziation within GGA. We use GGA for global optimization while varying the percentage g of offspring that are engineered and the percentage r of non-competitive individuals which are replaced by new randomly generated individuals.

Given two parent genomes, a child is normally created through a random process that mixes the AND-OR trees storing the parameter settings of each parent. To assess the effect of *perfect* genetic engineering (using a perfect surrogate model and perfect optimization), with a certain probability g we will replace this standard crossover operator with a method that *brute-force enumerates all possible combinations* of the two parents and evaluates them on the *actual objective function*.[3] This allows us to find the best possible child given a pair of parents.

To assess the importance of diversification in combination with highly greedy recombination, we replace a percentage r of the non-competitive population with completely randomly generated genomes. This is a common strategy in, e.g., random-key GA approaches [Bean, 1994].

We assess the impact of genetic engineering and increased diversification on the performance of GGA by using it for globally optimizing ten 4 dimensional and ten 8 dimensional black box optimization (BBO) functions as described in [Hansen *et al.*, 2009] from the COmparing Continuous Optimizers (COCO) software [Hansen *et al.*, 2011]. The test functions in COCO are highly non-convex and quick to evaluate, which provides a good initial testbed for evaluating different configurations of GGA.

We stress, however, that the BBO instances we consider differ from automatic algorithm configuration in several important ways: First, real algorithms usually have a mixed set

[3] We emphasize that due to the clearly exponential runtime of this routine we use this for investigative purposes only!

Population	g	r				
		0.0	0.25	0.5	0.75	1.0
50	0.00	0.301	0.308	0.329	0.363	0.385
	0.25	0.219	0.195	0.186	0.182	0.180
	0.50	0.186	0.159	0.154	0.151	0.148
	0.75	0.169	0.142	0.135	0.133	0.132
	1.00	0.156	0.129	0.124	0.123	0.120
75	0.00	0.230	0.247	0.281	0.316	0.339
	0.25	0.168	0.157	0.153	0.149	0.155
	0.50	0.146	0.134	0.128	0.126	0.123
	0.75	0.128	0.116	0.113	0.111	0.112
	1.00	0.121	0.106	0.106	0.102	0.104
100	0.00	0.193	0.222	0.255	0.289	0.318
	0.25	0.139	0.137	0.132	0.134	0.138
	0.50	0.119	0.113	0.111	0.113	0.114
	0.75	0.107	0.101	0.098	0.099	0.099
	1.00	0.098	0.092	0.092	0.092	0.093

Table 1: Average score across 100 GGA executions of functions 15 through 24 on dimensions 4 and 8 with varying g and r probabilities. The number of generations is fixed at 50.

of continuous, ordinal, and categorical parameters. Moreover, they frequently have tens, at times hundreds of parameters. Finally, in algorithm configuration we are often forced to approximate the performance of a target algorithm on a training benchmark for a given parameter setting by evaluating the algorithm on varying subsets of instances.

We run GGA to *minimize* BBO functions 15 – 24 with 4 and 8 dimensions 100 times for each function/dimension pair on different settings of g and r for different population sizes. Table 1 shows the results of this experiment. As the optimum value changes between functions and in some cases is even unknown, before averaging results between functions, we normalize the values between the best and worst observed result, with 0 being the best objective found and 1 the worst.

As shown in Table 1, when the percentage g of genetical engineering is set to 0, increasing the random replacement probability r worsens the results of GGA significantly (the top row of each population amount). This means that if GGA follows its original crossover approach to generate new children, randomly throwing out parameterizations from the non-competitive pool and replacing them with random parameterizations is ill advised. However, as soon as g is increased, the introduction of new random parameterizations is beneficial, likely helping to compensate the greater intensification by increasing the diversification of the non-competitive pool.

Most importantly, though, we observe that genetic engineering really boosts performance. Even without increasing r, increasing g to a point where *all* offspring is engineered results in roughly twice the performance quality compared to GGA's standard operator. While we had originally assumed that there was some mix of random and greedy crossover that would yield the best results, the results clearly show that, to maintain gene pool diversity, it is more favorable to inject new random individuals rather than using some fraction of random crossovers.

4.2 Configuring SAT Solvers

While encouraging, the experiments on BBO are not fully indicative of the typical scenarios where algorithm configuration is utilized. In practice, we do not have the luxury of perfectly engineering offspring. Moreover, the performance of the target solvers is much more erratic and can only be approximated through evaluations over multiple instances. Furthermore, the number of parameters that need to be set can span into the hundreds and vary among continuous, discrete and categorical. In this section, we therefore evaluate our proposed strategies on a real-world scenario, the tuning of two highly sophisticated SAT solvers: glucose 4.0 [Audemard and Simon, 2009] and lingeling (version ayv-86bf266) [Biere, 2014].

We choose these solvers for two main reasons. First, both are high quality parameterized solvers that have performed very well in SAT competitions. Secondly, we choose this pair because glucose only has 8 adjustable parameters while lingeling has over 100. This disparity allows us to test our approaches in both extremes.

We evaluate the solvers on a collection of industrial instances spanning the 2009 SAT Competition to the one in 2013. There are 727 instances total. For training, we evaluated each solver's default configurations on all instances to isolate the ones where the solver needed at least 30 seconds to solve an instance, but could also finish it in under 300 seconds. We ignore very easy instances as they can be handled by a pre-solver. We also ignore very hard instances because we want GGA to have a chance of solving an instance without wasting time on something it will never solve. From these subsets of instances we used 115 for training glucose, and 50 for training lingeling. Subsequently, *all* instances not used for training that could be solved by the default or any parametrization found by the various configurators, including both easy ones and those where the default parameters time out, were used for testing.

Using GGA the allotted training time was set to 48 hours, with the population size set to 100, the maximum number of generations to 50, and the tournament size to 8. Both tuning and evaluation are performed with a timeout of 300 seconds on AMD Opteron 6134 processors with 64GB of RAM. From the experiments with BBO we observed that engineering all offspring is beneficial when also diversifying the non-competitive pool, and therefore, based on Table 1, we set $g = 1$ and $r = 0.5$.

Surrogate Model Optimization: We compare three methods for surrogate-model-based genetic engineering (GE) and the original GGA. First, we evaluate an off-the-shelf random forest that randomly samples 100,000 possible children of any two given parent genomes and selects the one predicted to have the best performance, RF. Second, we evaluate a random forest that practices the focused sampling search based on the structure of the decision trees with 50 samples per leaf, Structured. Finally, we consider again the focused sampling approach but this time based on trees that were built giving higher resolution to the high-performance regions of the parameter space, Targeted. These results are summarized in Table 2.

The table compares the algorithms on five metrics: percentage of instances solved, average time needed on instances solved, median run time on instances solved, and the penalized average runtimes PAR1 and PAR10 where the timeout is included when computing the average time. In contrast

		GE			GE + Sexual Selection	
glucose	GGA	RF	Structured	Targeted	RF-S	Targeted-S
Solved (%)	92.5	88.8	91.2	93.7	91.9	92.5
Average	79.3	86.5	89.77	84.4	84.34	82.5
Median	47.2	55.8	57.8	49.8	57.3	52.8
PAR1	95.8	110.5	101.8	97.9	101.9	98.8
PAR10	298.3	414.3	344.4	266.7	321.2	301.3
		Varying Surrogate Models			GE + Sexual Selection	
lingeling	GGA	RF	Structured	Targeted	RF-S	Targeted-S
Solved (%)	89.8	72.2	82.4	93.5	92.6	90.7
Average	77.0	84.5	86.4	78.7	92.0	91.0
Median	44.2	62.8	60.2	50.3	76.8	74.8
PAR1	99.7	144.4	124.0	93.0	107.4	110.3
PAR10	374.7	894.4	599.0	268.0	307.4	360.3

Table 2: Comparison of varying GGA modifications for tuning glucose and lingeling configurations on SAT instances

to "Average", PAR1 counts timeouts but does not penalize them. PAR10 counts timeouts as if the solver needed 10 times the timeout. In the literature, the chief comparison is usually the percentage of instances solved and the PAR10 score. We therefore task all configurators to minimize PAR10.

Table 2 shows that employing an off-the-shelf random forest in combination with a random sampling optimization method does not improve the performance of GGA. Indeed, the opposite is true: GGA's performance deteriorates significantly. The same is true, albeit to a lesser extent, when using a structured sampling method based on an off-the-shelf random forest which aims at predicting well across the board.

However, if the underlying surrogate model is created to focus on areas of target function extrema, performance is markedly improved. For the glucose solver, which has few parameters, genetic engineering with the targeted surrogate model still achieves an improvement of 1% in the number of instances solved. On lingeling, which has many parameters, genetic engineering determines a configuration of lingeling that is able to solve about 4% more instances than the configuration GGA finds. Note that these two high-performance solvers have undergone rigorous manual tuning and therefore improving their performance further is challenging.

Sexual Selection: Having shown that a good surrogate model can offer noticeable improvements within GGA through genetic engineering, the question becomes whether sexual selection can give an additional boost to performance. Table 2 presents these results in the last two columns. Specifically, we first add sexual selection to the approach using an off-the-shelf random forest, RF-S. We also evaluate the performance of genetic engineering with a specially modified forest, Targeted-S.

In both benchmarks, the proposed sexual selection strategy on top of genetic engineering is detrimental compared to the "Targeted" genetic engineering method in isolation. After the BBO experiments showed that very aggressively engineering offspring at all times was the best choice under perfect conditions, this result came as a surprise. We believe that using a surrogate model to indirectly evaluate the non-competitive subpopulation de-facto creates competition in this part of the population and thus undermines its ability to serve as diversity store. More research is required to find out whether alternative methods for matching parents can work better.

State of the Art: We conclude our experiments by comparing both GGA and SMAC with GGA++, which replaces the randomized crossover operator with a genetic engineering approach based on a specially tailored random forests that focus on accurately predicting high-quality regions of the parameter space. GGA++ does not use sexual selection. For evaluating SMAC, we adjusted the wall-clock time to 48 hours (the time needed by GGA to conduct 50 generations), and tuned 8 versions of the configurator for each training set, running the best one on the final test data. In Table 3 we show the average test results when configuring glucose and lingeling ten times with each configurator.

As we observe, GGA++ shows improvements over the non-model-based GGA as well as the model-based SMAC. For glucose, GGA++ lowers the PAR10 score by over 5% compared to SMAC. For lingeling, which has significantly more parameters and thus offers more room for improvement, GGA++ lowers the PAR10 score by almost 20%.

The performances on the training sets, while naturally higher, followed a similar trend of GGA++ providing the best performance. Furthermore, for lingeling a mere ten repetitions are enough to claim statistical significance of the improvements achieved by GGA++ compared to both the default and SMAC. Note, however, that this significance is limited by the parameters of the experimentation, in particular the specific solver tuned and train-test split considered.

5 Conclusion

We showed how training a surrogate model can improve the performance of a gender-based genetic algorithm to create a more powerful algorithm configurator. Specifically, we showed how to integrate a surrogate model into the GGA configurator, by genetic engineering as well as sexual selection, whereby the former led to marked improvement whereby the latter was detrimental. We also developed a new surrogate model based on random forests that aims at higher resolution approximations where the target function exhibits good performance. Overall, this resulted in a new algorithm configurator that outperformed both a state-of-the-art non-model-based and a state-of-the-art model-based configurator when tuning two highly performant SAT solvers.

glucose	Default	GGA	SMAC	GGA++
Solved (%)	86.9	86.5	87.0	87.9
Average	87.7	84.7	85.5	85.3
Median	71.0	71.2	72.1	68.4
PAR1	115.4	114.0	113.5	111.4
PAR10	468.7	479.5	464.7	438.2
lingeling	Default	GGA	SMAC	GGA++
Solved (%)	84.0	84.1	82.1	87.8
Average	95.4	93.7	87.1	95.6
Median	106.6	97.7	85.8	82.3
PAR1	128.2	126.7	125.4	120.5
PAR10	560.7	556.8	608.8	449.3

Table 3: Average performance comparison of state-of the art configurators on SAT solvers glucose and lingeling

References

[Adenso-Diaz and Laguna, 2006] B. Adenso-Diaz and M. Laguna. Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research*, 54(1):99–114, 2006.

[Ansotegui *et al.*, 2009] C. Ansotegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, pages 142–157, 2009.

[Ansótegui *et al.*, 2014] C. Ansótegui, Y. Malitsky, and M. Sellmann. Maxsat by improved instance-specific algorithm configuration. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[Audemard and Simon, 2009] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 399–404, 2009.

[Ausiello *et al.*, 1999] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A .Marchetti Spaccamela, and M. Protasi. Complexity and approximation. *Combinatorial Optimization Problems and their Approximability Properties*, 1999.

[Bean, 1994] J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.

[Biere, 2014] A. Biere. Lingeling essentials, A tutorial on design and implementation aspects of the the SAT solver lingeling. In *POS-14. Fifth Pragmatics of SAT workshop, a workshop of the SAT 2014 conference, part of FLoC 2014 during the Vienna Summer of Logic, July 13, 2014, Vienna, Austria*, page 88, 2014.

[Birattari *et al.*, 2010] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated f-race: An overview. In *Empirical Methods for the Analysis of Optimization Algorithms*, pages 311–336, 2010.

[Box and Wilson, 1951] G.E.P. Box and K.B. Wilson. On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society Series*, 1951.

[Glover *et al.*, 2000] F. Glover, M. Laguna, and R. Marti. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

[Hansen *et al.*, 2009] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noisy functions definitions. *INRIA Research Report RR-6869*, 2009.

[Hansen *et al.*, 2011] N. Hansen, S. Finck, and R. Ros. Coco: Comparing continuos optimizers. Research report INRIA-00597334, INRIA, 2011.

[Hutter *et al.*, 2009] F. Hutter, H.H. Hoos, K. Leyton-Brown, and T. Stuetzle. Paramils: An automatic algorithm configuration framework. *JAIR*, 36:267–306, 2009.

[Hutter *et al.*, 2011] F. Hutter, H.H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, pages 507–523, 2011.

[Jin, 2011] Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61 – 70, 2011.

[Kadioglu and Sellmann, 2009] S. Kadioglu and M. Sellmann. Dialectic search. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, CP'09, pages 486–500, 2009.

[Kadioglu *et al.*, 2010] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. ISAC–Instance-Specific Algorithm Configuration. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 751–756. IOS Press, 2010.

[Malitsky *et al.*, 2013a] Y. Malitsky, D. Mehta, B. O'Sullivan, and H. Simonis. Tuning parameters of large neighborhood search for the machine reassignment problem. In *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems*, pages 176–192. Springer, 2013.

[Malitsky *et al.*, 2013b] Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Algorithm portfolios based on cost-sensitive hierarchical clustering. *IJCAI*, 2013.

[Rasheed and Hirsh, 2000] K. Rasheed and H. Hirsh. Informed operators: Speeding up genetic-algorithm-based design optimization using reduced models. In *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO*, pages 628–635. Morgan Kaufmann, 2000.

[Wedekind *et al.*, 1995] C. Wedekind, T. Seebeck, F. Bettens, and A.J. Paepke. Mhc-dependent mate preferences in humans. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 260(1359):245–249, 1995.

[Xu *et al.*, 2012] L. Xu, F. Hutter, J. Shen, H.H. Hoos, and K. Leyton-Brown. Satzilla2012: Improved algorithm selection based on cost-sensitive classification models, 2012. SAT Competition.