On Using Surrogates with Genetic Programming

Torsten Hildebrandt
Bremer Institut für Produktion und Logistik GmbH (BIBA) an der Universität Bremen, Bremen, Germany
hil@biba.uni-bremen.de

Jürgen Branke
Warwick Business School, University of Warwick, CV4 7AL Coventry, UK
Juergen.Branke@wbs.ac.uk

doi:10.1162/EVCO_a_00133

Abstract
One way to accelerate evolutionary algorithms with expensive fitness evaluations is to combine them with surrogate models. Surrogate models are efficiently computable approximations of the fitness function, derived by means of statistical or machine learning techniques from samples of fully evaluated solutions. But these models usually require a numerical representation, and therefore cannot be used with the tree representation of genetic programming (GP). In this paper, we present a new way to use surrogate models with GP. Rather than using the genotype directly as input to the surrogate model, we propose using a phenotypic characterization. This phenotypic characterization can be computed efficiently and allows us to define approximate measures of equivalence and similarity. Using a stochastic, dynamic job shop scenario as an example of simulation-based GP with an expensive fitness evaluation, we show how these ideas can be used to construct surrogate models and improve the convergence speed and solution quality of GP.

Keywords
Genetic programming, surrogates, phenotypic characterization.

1 Introduction
For many practical applications, the most time-consuming part of evolutionary algorithms (EAs) is the fitness evaluation. In engineering design, for example, evaluating a single solution may take several hours. In order to speed up the execution of EAs with expensive fitness evaluations, integrating surrogate functions has attracted much research (Jin, 2005, 2011). The basic idea here is to minimize the number of expensive fitness evaluations by replacing them with a computationally cheaper function as far as possible. To learn such surrogate functions from a sample of fully evaluated solutions, various statistical or machine learning techniques can be used. These models usually assume a numerical representation of the solutions, as they require a measure of distance or similarity between solutions. Because the genotype in genetic programming (GP) is a tree structure, it is not straightforward to use surrogate models with GP.

In this paper, we present an approach that allows surrogate models to be used with GP. Rather than trying to learn an approximate function that maps the genotype (i.e., a GP tree) into the fitness, we introduce the concept of a phenotypic characterization of a solution. This phenotypic characterization tries to capture the behavior of the

1 Sometimes surrogate functions are also called metamodels or response surface models.
T. Hildebrandt and J. Branke

phenotype encoded in the GP tree, and represents it as a vector of numerical values, which then allows calculating distances among individuals as required by the surrogate models. We show how this idea can be used to remove duplicates and reduce the number of necessary expensive fitness evaluations, thus generating better results more quickly. Our approach thus makes GP more efficient when applied to problems with expensive fitness evaluation, and opens up the possibility of using GP on problems which previously have been computationally too demanding.

The exact phenotypic characterization is problem specific. However, the general idea is generic and should be applicable across application domains. To demonstrate the usefulness of our proposal, we use the task of evolving dispatching rules for shop scheduling as an example of GP with expensive fitness evaluation.

This paper is structured as follows. After reviewing related work in Section 2 we describe our application area in Section 3. In Section 4, we introduce the concept of a phenotypic characterization, present such a characterization for dispatching rules, and describe its use to establish (approximate) phenotypic equivalence and similarity between GP trees encoding dispatching rules. Section 5 presents our experimental setup. The results presented in Section 6 demonstrate the usefulness of the introduced concepts to improve the convergence of GP by

1. detecting and suppressing phenotypic duplicates, and
2. supplementing GP with a surrogate function that uses approximate phenotypic similarity between individuals.

We analyze what data to base the phenotypic characterization on, and compare the performance of a genotypic similarity measure for GP trees with the phenotypic similarity introduced in this paper. We conclude with a summary and outlook toward future work.

2 Related Work

The use of approximate but cheap method to evaluate learned surrogate models can speed up EAs on problems with expensive fitness functions. It has been proposed many years ago (e.g., Rattle, 1998) and meanwhile is an established research area. In the following, we briefly discuss some of the most relevant papers. (For a more comprehensive review, the interested reader is referred to, e.g., Jin, 2005, 2011; Shi and Rasheed, 2010; Jin and Branke, 2005.)

The surrogate model is usually learned from samples of expensive evaluations. Popular choices for surrogate models include neural networks (Hong et al., 2003), regression techniques (linear or polynomial; Oduguwa and Roy, 2002; Branke and Schmidt, 2003), nearest neighbor interpolation (Runarsson, 2004), or Gaussian processes (El-Beltagy et al., 1999). Recently, approaches were also proposed that combine different models (possibly using different techniques) or select the most promising model(s) depending on the current situation (see, e.g., Goel et al., 2007; Lim et al., 2007; Le et al., 2013). The data used to learn the surrogate model can come from an experimental design conducted before the EA, but usually is learned and adopted online during the run (Jin, 2005, 2011; Jin and Branke, 2005).

Surrogate models can be used to support EAs in several ways. Most often, all individuals are evaluated with the surrogate model, and this information is used to preselect the most promising individuals that are then evaluated with the expensive fitness function (Jin et al., 2002). That way, individuals classified as poor by the surrogate model can be quickly discarded, while the better individuals are fully evaluated to allow
On Using Surrogates with Genetic Programming

an accurate ranking. Furthermore, the full evaluations of these individuals are used to
further refine the surrogate model particularly in the most promising areas. Others have
proposed to alternate between generations that are evaluated with the surrogate model
and generations that are evaluated using the expensive fitness function (Rattle, 1998;
Jin et al., 2002), to use surrogate models for supporting local search (Lim et al., 2010), or
for preselection among individuals generated by a genetic operator (Emmerich et al.,
2002).

However, so far, successful use of surrogates to speed up EAs has been mostly re-
stricted to fixed-length representations for which it was easy to define a distance metric
(e.g., bit strings or vectors of real values). GP, on the other hand, usually uses a tree
representation of variable length. Machine learning or statistical techniques cannot be
used directly to work with such genotypes or phenotypes, which probably is the reason
why, to the best of our knowledge, only recently work on using surrogate functions with
GP has been reported (Kim et al., 2014). They generalize radial basis function networks
so they can use arbitrary distance metrics and thus be used in combinatorial or program
spaces. Kim et al. (2014) also report on initial experiments using GP with these RBF-
based surrogates. Results for very small computational budgets using the structural
Hamming distance between trees (see Sections 5.3 and 6.3) for a unimodal problem, the
four-odd parity problem, and a symbolic regression problem show improved results
over random search and standard GP. As an alternative to their approach, we overcome
the difficulties of applying surrogate models with GP by using an (approximate) phe-
notypic characterization of the individual, instead of working on a syntactic level. As it
is a vector consisting of integer values, we can then apply standard techniques from the
surrogate literature to estimate the fitness value of a GP individual without performing
an expensive fitness evaluation.

A few papers in the GP area are related to our idea of phenotypic representation
in the sense that they propose considering the behavior of solutions in GP rather than
just the genotype. So far, however, this idea has not been used to construct surrogate
models.

Gustafson (2004) indicates that there is a positive correlation between diversity and
performance achieved by GP for many problems, and consequently high diversity is
considered beneficial. A typical approach to maintaining diversity is fitness sharing.
It forces similar individuals to share their fitness values, and thereby discourages the
formation of clusters of very similar individuals. This requires a similarity metric. While
similarity between GP trees is usually calculated using syntactic similarity (Ekärt and
Németh, 2000), recent research by Nguyen et al. (2012) aims at incorporating sampling
semantics to form clusters sharing their fitnesses. This sampling semantics is in some
respects similar to our phenotypic characterization. Jackson (2011) modified GP elim-
inating candidate solutions with duplicate objective function values in the initial pop-
ulation as well as during the GP run and reported improved performance. Detecting
such duplicates by comparing objective values is quite costly in such a simulation-
opimization context as ours, as full simulation runs are required to obtain objective
function values. The amount of duplicates can be reduced to a large extent without
additional simulations, however. Branke et al. (2014) used a simplified version of the
phenotypic characterization to cheaply compute approximate phenotypic equivalence
between trees and could demonstrate its usefulness to improve convergence speed and
solution quality of standard GP. Here we further analyze this approach and extend it to
create surrogate functions to improve the convergence of GP. As an extreme option to
diversity preservation, Lehman and Stanley (2010) propose focusing on novelty instead
of the objective function value when selecting individuals, and again use the phenotypic behavior to determine novelty.

In Uy et al. (2011) the authors use the abovementioned sampling semantic distance in order to define tree similarity and use it in an improved crossover operator to enforce crossover between subtrees in a certain similarity range. They report improved results on a number of symbolic regression problems.

Ashlock and Lee (2013) recently proposed the concept of agent-case embedding, which is closely related to our concept of phenotypic characterization and is used there for analyzing evolved systems.

Finally, the work reported in this paper is somewhat related to the area of model-based GP. Transferring the ideas of estimation of distribution algorithms (see, e.g., Larrañaga et al., 2012) to GP, a probabilistic model is used in these approaches to capture knowledge about high-quality areas of the search space. Replacing standard GP operators for selection, crossover, and mutation completely, this model can be used for creating new individuals by sampling. However, unlike a surrogate model, an EDA-model typically does not predict fitness values, whereas a surrogate model is not constructive, that is, it cannot be used to easily sample new candidate individuals.

3 Evolving Dispatching Rules

The application we consider in this paper as an example is the problem of evolving dispatching rules for dynamic job shop scheduling problems. Job shop scheduling (Pinedo, 2008) tries to optimize the assignment of tasks (operations of a job) to machines in order to optimize a certain objective function such as the mean flowtime or the mean tardiness of jobs. There are technological (operations of a job with certain time requirements have to be completed in a certain order) as well as capacity constraints (each machine can process at most one operation at a time). Once an operation is started, it cannot be interrupted (nonpreemptive). Job shops are different from other shop configurations in allowing a very flexible job configuration. Each job can have a different routing and different processing time requirements. In contrast to, for example, flow shops, this can used to model one-of-a-kind or small-series production. Job shop scheduling is a NP-hard problem (Pinedo, 2008).

A dispatching rule is a simple heuristic that computes a priority value for a job from its attributes (Haupt, 1989; Blackstone et al., 1982; Holthaus and Rajendran, 1997; Rajendran and Holthaus, 1999). Whenever a machine becomes idle and has waiting operations in its queue, the job with the highest priority value is chosen to be processed next. Benefits of dispatching rules include their simple and intuitive nature, their ease of implementation within practical settings, and their flexibility to incorporate domain knowledge and expertise. They are very well suited for online scheduling, as they automatically take into account the latest information available. All these factors have led to their frequent use in various industries, including semiconductor manufacturing (Pfund et al., 2006).

Usually dispatching rules are designed by experts in a tedious trial-and-error process, with candidate rules being tested in a simulation environment, and them modified and re-tested until they fulfill the requirements for actual implementation in practice (Geiger et al., 2006). In recent years, several authors have suggested automatically designing suitable dispatching rules by means of EAs (see e.g., Geiger et al., 2006; Dimopoulos and Zalzala, 2001; Pickardt et al., 2013, 2010; Nguyen et al., 2013, 2014). Most of these approaches use GP (Koza, 1992; Poli et al., 2008) and can be classified
On Using Surrogates with Genetic Programming

as applications of hyperheuristic search (Burke et al., 2009, 2010). As such, their search space is the space of heuristics instead of a concrete solution of a problem instance.

In terms of the set of terminals and nonterminals, finding a dispatching rule is similar to symbolic regression, a standard GP benchmark problem. A GP individual encodes an arithmetic expression, a function of several variables. In case of a dispatching rule, the GP individual encodes a function \( p : \mathbb{R}^a \rightarrow \mathbb{R} \) mapping a vector of \( a \) attributes describing a job, machine, or the whole system to a priority value. Fitness evaluation, however, is far more complex than the case of symbolic regression, because normative data (i.e., what priority value to produce for a certain set of attribute values) is not available. Thus, fitness evaluation usually involves running multiple replications of a stochastic, dynamic shop floor simulation. The time required for just a single fitness evaluation can therefore easily exceed the computational time required by the core GP algorithm.

In recent years, GP was used to successfully develop improved dispatching rules for various scheduling problems ranging from single machine problems (e.g., Geiger et al., 2006), classic static job shop scheduling problems (Nguyen et al., 2013), to dynamic job shop scheduling problems (e.g., Hildebrandt et al., 2010). Recently also dispatching rules for a complex dynamic scenario from semiconductor manufacturing were evolved using GP (Pickardt et al., 2013, 2010). This scenario consists of 36 machines, sequence dependent setup times, parallel machines, batch machines, and reentrant process flows with jobs consisting of up to 92 operations. Rules developed by GP for this scenario clearly outperformed standard dispatching rules from the literature. To use GP in such a setting is computationally very demanding. The method presented in this paper is one step toward an increased scalability of GP to even larger practical scenarios requiring even longer simulation times.

In terms of the scheduling scenario considered, we take one step back in this paper by considering a dynamic job shop scenario with 10 machines proposed by Rajendran and Holthaus (1999) and described in more detail in Section 5.1. Using this scenario (taking about 0.6 s per fitness evaluation) seems complex enough to justify the use of a surrogate model, but still keeps computational time low enough to be able to investigate a variety of different parameter settings and perform each experiment a number of times to obtain statistically valid results on mean performance.

A good benchmark rule for minimizing flowtime in this scenario is the rule manually developed by Holthaus and Rajendran (2000). In this rule, the priority of a job is calculated as \(-2(PT + WINQ + NPT)\), and is thus a linear combination of the current processing time (PT), work content in next queue (WINQ), and the processing time of a job’s next operation (NPT—next processing time). In the following, we call this rule the Holthaus rule and use it as the reference rule throughout the paper, as it is the best manually developed rule we know of for the scheduling scenario considered in this paper. An example of a GP expression tree for the Holthaus rule is depicted in Figure 1.

4 Approach

4.1 Overview

Before presenting in detail how to compute the phenotypic characterization and how to use it to define (approximate) equivalence and similarity, we show in this section how they are embedded in the overall solution procedure. Figure 2 depicts the general outline of an optimization run. The general procedure is based on a generational reproduction scheme, as used for instance in conventional GP. In Figure 2 standard steps are shown.
T. Hildebrandt and J. Branke

Figure 1: GP tree representation of the Holthaus rule – (2PT + WINQ + NPT) (Holthaus and Rajendran, 2000), where PT, WINQ, and NPT denote processing time, work in next queue, and processing time at the next machine, respectively.

Figure 2: General solution approach. Components of standard GP are shown as white boxes. Additional steps related to the use of surrogates and detection/removal of phenotypic duplicates are shown as gray boxes.

as white boxes, additional steps related to the use of surrogates and detection/removal of phenotypic duplicates are shown as gray boxes. Each iteration of the main loop (steps 3 to 9) constitutes one generation.

We start with a population $P$ of rules/individuals (step 1). These initial rules are usually randomly generated using some initialization procedure. In step 2 we remove duplicate rules using the procedure described in Section 4.3 and replace them with additional random rules until our population consists of the required number of different rules.

All candidate rules in $P$ are subsequently fully evaluated in a simulation in step 3. As already mentioned, this is by far the most time-consuming step in the overall procedure. Once real fitness values are known, we check whether the optimization run should be terminated or continue with another iteration. If we decide to terminate, the best rule found so far is reported as the result of the optimization run. Otherwise, we...
update our surrogate model to incorporate the new fitness values of all individuals in $P$ (step 4).

In the next step (step 5), conventional GP operators (see Koza, 1992; Poli et al., 2008) for selection, mutation, and crossover are used to produce a new, intermediate population $P_{\text{imd}}$ taking into account all individuals in $P$ and their fitness values. Tree mutation works by replacing a node of the tree by a new, randomly created subtree. Tree crossover takes two parent trees and produces two offspring trees by exchanging a randomly selected subtree between the two parents.

Duplicates in $P_{\text{imd}}$ are subsequently removed in step 6 and replaced with additional individuals generated from $P$ in the same way as in step 5. At this stage we have a set of individuals which produce with a very high probability distinct (as will be examined in Figure 4(a)) fitness values.

In conventional GP, both $P$ and $P_{\text{imd}}$ contain the same number of individuals. As we want to use the surrogate model to select the most promising individuals later on, we have to create more rules in this step, that is, $|P_{\text{imd}}| = n|P|$ with $n > 1$. In the experimental section of this paper, we investigate different values of $n$. We only use integer values for the experiments in this paper, but generally any real number $>1$ could be used. Setting $n$ too low will result in only minor improvements over standard GP, and setting it too large may have negative consequences on convergence, because inaccuracies of the surrogate function may have a lot of impact and misguide the GP.

In the next step (step 7), we determine the phenotypic characterization for all rules in $P_{\text{imd}}$ following the procedure explained in the next section. This is used to approximate the fitness values for all individuals using the surrogate model in step 8. Computing the phenotypic characterization of a rule and evaluating the surrogate for it once is computationally far less demanding than a full fitness evaluation.

As a next step (step 9), estimated fitnesses are used to select the most promising candidates to form a new population $P$ for the next iteration. Step 3 starts a new iteration of the main optimization loop again, fully evaluating each individual in $P$ using the expensive fitness function.

4.2 Phenotypic Characterization of Dispatching Rules

The key to using a surrogate model with GP is to come up with a meaningful distance metric. Because this is difficult on the genotype level (trees), in the following we propose a phenotypic characterization. The phenotypic characterization looks at the behavior of the evolved rules, rather than their syntactic description. This is naturally problem-dependent. In the following, we develop a phenotypic characterization for evolving dispatching rules in job shop scheduling. However, similar phenotypic characterizations can also be developed for other problem domains, and Ashlock and Lee (2013) may serve as additional inspiration.

Dispatching rules define an ordering relation among jobs. Each time a machine is free and there are jobs waiting, the rule is used to sequence all jobs by their priority, and the job with highest priority is processed next. The key characteristic of dispatching rules is not the functional values they compute, but only to which of a set of jobs they assign the highest priority.

Consequently, the phenotypic characterization for dispatching rules we propose here is a decision vector, a record of all the decisions a dispatching rule takes in a comparably small number of $k$ decision situations, as outlined in Figure 3. These decision situations could be sampled from an actual simulation run, or simply be created randomly. This latter aspect is analyzed in more depth in Section 6.2.
T. Hildebrandt and J. Branke

<table>
<thead>
<tr>
<th>decision situation</th>
<th>attribute set s</th>
<th>ranking by reference rule</th>
<th>ranking by other rule</th>
<th>decision vector d</th>
<th>fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s1 4 8</td>
<td>1</td>
<td>2</td>
<td>d1   2</td>
<td>1456</td>
</tr>
<tr>
<td>2</td>
<td>s2 17 1</td>
<td>2</td>
<td>1</td>
<td>d2   1</td>
<td>1123</td>
</tr>
<tr>
<td>k</td>
<td>s3 9</td>
<td>5</td>
<td>3</td>
<td>d3   3</td>
<td>1293</td>
</tr>
</tbody>
</table>

(a) Figure 3: Phenotypic characterization of dispatching rules. (a) Creation of a decision vector for a certain rule given a set of k decision situations. Columns 3 and 4 show arbitrary examples of attribute values, columns 5–7 are job indices selecting a job of their respective decision situation. (b) Database of m rules, their decision vectors and fitness values.

Algorithm 1 Compute the phenotypic characterization

Input: rnew: the dispatching rule to characterize  
Input: rref: the reference rule  
Input: S: set of |S| decision situations

Output: d: decision vector with |S| elements
1: d ← new integer vector with |S| elements
2: for i ← 1, |S| do
3: s ← S[i]  
4: p_ref ← apply(r_ref, s)  
5: k_ref ← ranks(p_ref)  
6: p_new ← apply(r_new, s)  
7: k_new ← ranks(p_new)  
8: j ← arg min(k_new)  
9: d[i] ← k_ref[j]  
10: end for
11: return d

The rule R to be characterized is used to rank all jobs in each decision situation to find out which job is given the highest priority in each situation (rank 1). A certain fixed reference rule is also used to rank jobs in each decision situation. The decision vector d to characterize a rule R consists of, for each decision situation, the rank the reference rule assigned to the job receiving top priority by rule R. This is described more formally in the pseudocode of Algorithm 1.

Consider the example in Figure 3(a). In the first decision situation, there are two jobs to choose from. Rule R gave the highest priority (rank 1) to the second job. This job was ranked second by the reference rule, so d_1 = 2. Similarly, in the second decision situation, rule R gives the first rank to the second of three jobs, which has been ranked third by the reference rule, thus d_2 = 3. For decision situation k, rule R prioritizes the second job, which is in concordance with the reference rule, and thus d_k = 1.

In summary, d consists of k integer values expressing the similarity of R to the reference rule. A rule leading to the same decisions as the reference rule has all elements of the vector set to 1. A rule which always selects the job that has the lowest priority
according to the reference rule has all elements set to the highest possible values, that is, the number of jobs in the corresponding decision situation (note that this means the maximal error a rule can make depends on the number of jobs in a particular decision situation, and may be different for different situations).

This definition completely abstracts from the actual priority values produced by a rule. Therefore also syntactically quite distinct rules (such as “1/PT” vs. “−PT” as two possibilities to express the well-known rule “shortest processing time first”) would be recognized as being equivalent.

Obviously, the computational effort to compute the phenotypic characterization is much less than running a full, expensive fitness-evaluation, which would involve tens of thousands of such decisions.

4.3 Approximate Phenotypic Equivalence and Similarity

We use the phenotypic characterization to build a surrogate function that can help us estimate the fitness value of new individuals at much lower computational cost than the full evaluation via simulation. The surrogate function takes the phenotypic characterization (i.e., the decision vector \( d \) in our case) as input, and provides the estimated fitness as output. Constructing the surrogate function from a given set of input-output pairs (see Figure 3(b)) is a typical machine learning or regression task, and standard techniques such as neural networks, Gaussian processes, or regression trees could be used.

In this paper, we use the simplest possible regression model, the nearest neighbor regression. To predict the fitness of a new individual, the distances to all individuals in the database are computed using the Euclidean distance between phenotypic characterizations. The fitness of the most similar individual is returned as the fitness estimate for the new individual.

To detect and remove phenotypically identical rules, we could use identical decision vectors as a criterion. For this task, an even simpler criterion turned out to be sufficient, however: We only consider a single decision situation consisting of 100 jobs. Values of the attribute set are created randomly within typical value ranges but not taking into account any dependencies or correlations between the attributes. Two rules are considered identical if they produce the same rank vector for these 100 jobs.

5 Experimental Setup

5.1 Shop Floor Scenario

Our computational experiments use the dynamic job shop scenarios from Rajendran and Holthaus (1999) and try to find good dispatching rules for these scenarios automatically. For the particular problem setting we consider in this paper, there are 10 machines, each job entering the system is assigned a random routing, that is, machine visitation order is random with no machine being revisited. Processing times are drawn from a uniform discrete distribution ranging from 1 to 49 min. As we consider a dynamic problem, we simulate the system for a longer time span. Job arrival is a Poisson process, that is, interarrival times are exponentially distributed. The mean of this distribution is chosen to reach a desired utilization level on all machines. All experiments in this paper use a (mean) utilization of 95%. These scenarios were also used to evolve dispatching rules in Branke et al. (2014) and Hildebrandt et al. (2010).

Following the procedure from Rajendran and Holthaus (1999), we start with an empty shop and simulate the system until the first 2,500 jobs have been completed.
Data on the first 500 jobs is discarded to focus on the shop’s steady-state behavior. The objective function is to minimize the mean flowtime, that is, the arithmetic mean of the flowtimes of jobs 501 to 2,500: $FT = \frac{1}{2000} \sum_{j=501}^{2500} (C_j - r_j)$, where $C_j$ is the completion time of a job $j$ and $r_j$ is its arrival time.

Because the simulation is stochastic, the resulting flowtime is also a random variable. To reduce the impact of stochasticity, we replicate each simulation 10 times for each fitness evaluation and use the average performance for optimization. Furthermore, we use the same 10 random number seeds to evaluate all rules during optimization (common random numbers; Law 2007). Fixing the seed of the job shop simulation has the effect of fixing a certain problem instance, that is, a concrete set of jobs with a specific routing and specific processing times. All results reported in this paper are based on evaluating the resulting rules on a different set of 100 independent test runs, so in total 110 specific problem instances are used for the job shop simulations in this paper. The simulation has been implemented in Java, and is publicly available (Hildebrandt, 2012).

Most of the experiments described below use as the reference dispatching rule the manually developed Holthaus rule (Holthaus and Rajendran, 2000) described in Section 3. This rule has been shown to perform very well for the objective of minimizing mean flowtime.

### 5.2 Genetic Programming

For our experiments we used the GP implementation of the Evolutionary Computation in Java (ECJ)-framework. Details of the parameters used are shown in Tables 1 and 2. This choice of parameters and terminal set resembles the best settings identified for the tree representation in Branke et al. (2014), where we could also identify normalization of input attributes in a common value range of [0, 2] to improve GP performance. We therefore also use these settings for the experiments reported in this paper. Unless mentioned otherwise, convergence curves in this paper are based on 100 independent runs.

---

On Using Surrogates with Genetic Programming

Table 2: Information available to be used in dispatching rules (terminal set).

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Normal range</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT</td>
<td>[1, 47]</td>
<td>Processing time of current operation</td>
</tr>
<tr>
<td>NPT</td>
<td>[0, 47]</td>
<td>Next processing time, that is, processing time of next operation</td>
</tr>
<tr>
<td>WINQ</td>
<td>[0, 410]</td>
<td>Work in next queue, that is, sum of processing times of all jobs at the next machine</td>
</tr>
<tr>
<td>RemProcTime</td>
<td>[1, 264]</td>
<td>Remaining processing time, that is, sum of processing times of all unfinished operations</td>
</tr>
<tr>
<td>OpsLeft</td>
<td>[1, 10]</td>
<td>Operations left, that is, number of all unfinished operations</td>
</tr>
<tr>
<td>TimeInQueue</td>
<td>[0, 1500]</td>
<td>Time since arrival in current queue</td>
</tr>
<tr>
<td>TimeInSystem</td>
<td>[0, 2770]</td>
<td>Time since arrival in system</td>
</tr>
</tbody>
</table>

5.3 Surrogate Function

We use a nearest neighbor learner as a modeling technique. The nearest neighbor surrogate model uses the full fitness evaluations of the last two generations, that is, 1,000 fully evaluated individuals are used to predict the fitness of new individuals. To predict the fitness of a new individual, the distances to all the 1,000 individuals in the database are computed using the genotypic (syntactic) or phenotypic distance measures as described below. The fitness of the most similar individual is returned as the fitness estimate for the new individual.

As mentioned in the previous sections, it is very difficult to use surrogate functions for GP’s tree representation, as popular surrogate modeling techniques such as neural networks or polynomial regression can only learn functional relationships between real- or integer-valued vectors and a fitness value. In order to use a nearest neighbor learner, however, we only have to specify a distance measure between solutions. We can therefore use this learning method to compare surrogates based not only on the phenotypic characterization proposed above in Section 4.2, but also based on a genotypic distance metric from the literature.

More formally, for our phenotypic characterization, the distance $D$ between two decision vectors $d^A$ and $d^B$ with $k$ dimensions is computed as

$$D(d^A, d^B) = \sqrt{\sum_{i=1}^{k} (d^A_i - d^B_i)^2}$$

Therefore, the phenotypic distance $PD$ between two trees $T_1$ and $T_2$ can be calculated as $PD(T_1, T_2) = D(\text{phenoChar}(T_1, r_{ref}, S), \text{phenoChar}(T_2, r_{ref}, S))$, where $\text{phenoChar}(T, r_{ref}, S)$ computes the phenotypic characterization of a tree $T$ as described in Algorithm 1, using a reference rule $r_{ref}$ and a set of decision situations $S$.

Genotypic similarity between expression trees is computed using the structural Hamming distance (SHD), which was presented in Moraglio and Poli (2005) to create an improved crossover operator.

$$\text{SHD}(T_1, T_2) = \begin{cases} 
1 & \text{if } \text{arity}(p) \neq \text{arity}(q) \\
hd(p, q) & \text{if } \text{arity}(p) = \text{arity}(q) = 0 \\
\frac{1}{m+1} \left( \hd(p, q) + \sum_{i=1}^{m} \text{SHD}(s_i, t_i) \right) & \text{if } \text{arity}(p) = \text{arity}(q) = m
\end{cases}$$
It defines a syntactic distance measure between trees ranging from 0 (trees are equal) to a maximum distance of 1. In this formula $T_1$ and $T_2$ are trees, $p$ and $q$ are their root nodes, and $s_i$ as well as $t_i$ denote the $i$th subtree of $p$ and $q$, respectively. The utility function $\text{arity}(p)$ computes the number of a node’s subtrees. $\text{hd}(p, q)$ computes the Hamming distance between $p$ and $q$, which is 0 if $p = q$, that is, $p$ and $q$ represent the same type of terminal or nonterminal node, or 1 otherwise.

6 Results

6.1 Duplicate Detection and Removal

First, we look at the benefit of duplicate avoidance based on phenotypic characterization. As explained in Section 4.3, this is using the simplified phenotypic characterization based on the ranking of 100 artificial jobs. Based on this approximate equivalence we eliminate all duplicate rules in a population before running any simulations to assess rule performances. These rules are replaced with new rules either created randomly (initial population), or, later on in a GP run, by using the mutation/crossover operators.

As shown by the dotted line in Figure 4(a), this duplicate avoidance scheme is quite successful at maintaining a high level of diversity (measured in the number of individuals with distinct fitness values). Standard GP (solid line) starts with a considerably lower diversity, and depicts a further decreasing diversity after generation 13. The number of individuals created additionally to replace duplicates is shown in

Figure 4: (a) The fraction of individuals with unique fitness values for standard GP (solid line) and when using (approximate) duplicate detection (dotted line). (b) The number of additional individuals required to increase diversity (population size is 500). (c) Mean performance and (d) standard error of GP runs with (dotted lines) and without (solid lines) suppression of duplicates based on (approximate) phenotypic equivalence. Performance is relative to the mean flowtime achieved by the Holthaus rule and averaged over 100 runs.
Figure 4(b). Interestingly, especially for the initial population and the first few generations, the number of additionally created individuals is rather high, before dropping to a level of around 60 additional individuals per generation. From around generation 10 this number is increasing again with a linear trend. This is consistent with the behavior of standard GP: from about generation 10–13 it seems to become increasingly difficult for GP to find new, fitness-distinct individuals, therefore more individuals have to be created to compensate.

An increase in fitness diversity alone does not necessarily lead to good performance. Genotypically different individuals with the same fitness might represent different ways to express a certain dispatching rule, and preserving this information might be beneficial for the evolutionary process. However, a look at the convergence lines in Figure 4(c) shows a clear advantage of removing fitness duplicates: Both convergence speed and final solution quality improve. Also, the standard error and therefore standard deviation of run performance decreases (see Figure 4(d)), leading to more reliable optimization results. We can therefore conclude that at least in the considered application, removing phenotypically equivalent rules allows finding better solutions more quickly.

Summarizing, without using any additional expensive simulation runs, and with very low additional computational cost, we can almost entirely remove fitness duplicates, allowing a faster convergence to a better performance level.

6.2 Choosing a Set of Decision Situations

The phenotypic characterization of a dispatching rule (Section 4.2) depends on the rule’s behavior on a set of decision situations. Obviously, the set of decision situations selected has an impact on the quality of the resulting surrogate model, and the question arises how to create this set. For example, it could be created randomly, or by sampling real situations encountered in a simulation. To examine the influence of different ways to chose the set of decision situations, we analyze the ranking performance of the resulting surrogate models, based on the individuals of 30 GP runs (with suppression of duplicates but not using a surrogate function). Data of all individuals created is recorded and used to learn surrogate models from three different sets of decision situations; in all cases a set consists of 100 situations:

- **Random.** Decision situations are created randomly containing between 2 and 20 jobs. Similar to the technique to detect duplicates for each of these jobs, random values are created for each of the attributes within the ranges listed in Table 2 using a uniform continuous distribution. This approach assumes all attributes to be independent. It is pretty simple as it only depends on knowing typical value ranges of the attributes, no additional simulation runs are required.

- **Reference Rule.** Decision situations are sampled from preliminary simulation runs using the Holthaus rule. From all real decision situations encountered in such a simulation, a subset of 100 situations is sampled.

- **Optimized Rule.** As optimization progresses and decisions of optimized rules get increasingly different, situation characteristics might change. Thus, it might be advantageous to change decision situations during an optimization run. To quantify this effect we used the best rules from 30 GP runs to sample decision situations, anticipating the situations encountered by rules at the end of an optimization run. Obviously, this would not be feasible in practice, but may
serve as an indication for the performance when the set of decision situations is adapted dynamically to always reflect the best solution found so far.

We created 30 different random sets of decision situations and also sampled 30 different sets using the Holthaus rule. To assess the influence of optimized rules we used the best rule from each of the 30 recorded runs and sample a single set of decision situations with each one of them. Thus, for each setting we tested 30 different sets of decision situations to base the decision vector on. Using the nearest neighbor learner with Euclidean distance as a similarity measure, we created surrogate models using the phenotypic characterization based on each set of decision situations.

The quality of a surrogate model is judged as its ability to produce the same ranking that would result from the expensive fitness evaluations, which seemed the most relevant aspect given that we are using a rank-based selection scheme. We measure this by looking at the mean rank error of the predictions, normalized to the expected rank error of a random ordering of the individuals. The best possible error of 0 would mean the surrogate can produce a perfect ranking. In this case, the computationally expensive simulation runs could be replaced with the surrogate without any impact on GP performance. In the other extreme, a model with quality 1 is not better than using a random permutation of individuals. For a more detailed discussion of appropriate measures of surrogate quality see Jin et al. (2003).

The results in Figure 5 show the surrogate’s quality when using the individuals (and their fitness) from a certain generation as training data to predict the performance of the individuals of the next generation. For each curve in Figure 5 we tested each of the 30 different sets of decision situations with each of the 30 recorded runs.

The curves in Figure 5 show surrogate evaluation to be most accurate in the first few generations, probably because there is a high diversity in early generations, which makes it easier to distinguish between good and bad solutions. Surrogate quality decreases afterward until generation 30, and then mostly stagnates.
Comparing the three different settings to create decision situations, we achieve the best results using real decision situations from simulation runs (solid and dotted lines). Surrogates based on decision situations generated with optimized rules (solid) seem slightly better than if the reference rule is used (dotted). Though not immediately visible from the curves, performance differences between the two are statistically significant, except for the first two generations ($p < 0.001$ according to a nonparametric, paired Wilcoxon signed-rank test). But this difference is small compared to using decision situations that have been randomly generated, which performs much worse (dash-dotted line).

In the remainder of this paper, we therefore use situations sampled with the reference rule. Changing the set of decision situations dynamically during an optimization run can probably be used to further improve surrogate quality slightly, but is left for future work.

### 6.3 Phenotypic versus Genotypic Surrogate

The nearest neighbor surrogate model used in this paper only requires a distance metric to be applied. In order to demonstrate that the proposed phenotypic characterization allows for a much more meaningful distance metric than genotypic distances, in this section we compare the surrogate quality based on our phenotypic characterization with a surrogate model using the genotypic SHD distance measure (Moraglio and Poli, 2005), following the same experimental setup as described in the previous section.

Results are shown in Figure 6 comparing surrogate quality using phenotypic similarity from the previous section (based on random decision situations: dashed line; based on reference rule: dotted line) with genotypic similarity (dash-dot line). Results clearly show a much better performance of the surrogate model based on our phenotypic characterization. Even using random decision situations leads to much better results than the surrogate based on SHD distances. Especially in later generations, the ranking produced by the surrogate model based on SHD distances is only slightly better than simply creating a random permutation of individuals.
6.4 Phenotypic Surrogate for GP

In the previous two sections, surrogates were used to predict performance of GP individuals from recorded GP runs. We now report results from using surrogates dynamically in an actual GP run. Based on the results from the previous sections, we use decision situations sampled from simulations with the reference rule as the basis of the phenotypic characterization and predict rule performance using the nearest neighbor learner using Euclidean distance. As mentioned before, we use a preselection scheme to improve GP convergence by creating an intermediate population which contains a multiple $n$ of the population size of new individuals. Using the surrogate model based on the phenotypic characterization, the most promising fraction $1/n$ of individuals is selected to form the new population. This new population is fully evaluated using the simulation model. We investigate three different settings for the size of the intermediate population $n = 2, 5, 10$; the setting $n = 1$ does not use the surrogate model but only removal of duplicates.

The results are given in Figure 7(a). The solid and dotted lines show the results of standard GP and when using duplicate removal (as already presented in Figure 4(c)). Dashed lines show the convergence lines with surrogates. Different settings of $n$ are indicated by different shapes to mark data points (circle, square, diamond: $n = 2, 5, 10$). As can be seen, using the surrogate model results in faster convergence. Concerning the right size of the intermediate population there is a benefit of using larger values of $n$. Increasing $n$ shows decreasing returns, however. The value of the surrogate model therefore seems limited, and for larger $n$ the computational effort to create and evaluate (using the surrogate) additional individuals would only lead to marginally faster convergence.

These results are presented in a slightly different form in Figure 7(b) graphically showing the percentage of fitness evaluations saved while still achieving the performance of standard GP. Data for this graph was obtained by moving a horizontal line in Figure 7(a) along the solid line, that is, the convergence curve of standard GP. As an example, after 5,000 fitness evaluations (dotted vertical line next to the center of Figure 7(a)) standard GP achieves a performance of about 0.93. The same performance levels are achieved by the other variants considerably earlier, resulting in the saved budget values for a standard GP budget of 5,000 as shown in Figure 7(b). The final performance level of standard GP after 30,000 fitness evaluations is reached by the surrogate-assisted variants much earlier, resulting in savings of up to 80% of the fitness evaluations. Already GP with removal of duplicates reaches a very good performance level much more quickly. However, there is a clear additional benefit of using the surrogate model.

The convergence curves in Figure 7(a) also show another effect of increasing $n$: for large computational budgets, solutions without using the surrogate seem to converge to slightly better solutions in the end. To investigate this effect, we further analyzed the situation for a budget of 30,000 fitness evaluations, that is, at the rightmost data point in Figure 7(a). Figure 8(c) shows a boxplot of run performances for this budget and a detailed comparison of performance differences. The absolute performance differences in Figure 8(d) show mean performance differences (mean flowtime is measured in minutes) between the settings indicated by the methods in the corresponding row and column. For example, a difference of 10.2 between “standard” and “$n = 1$” means a worse performance of standard GP in comparison with the setting of $n = 1$ by 10.2 min of flowtime on average. Indications of statistical significance of these differences based on a nonparametric, paired Wilcoxon signed-rank test are shown in parentheses.
On Using Surrogates with Genetic Programming

Figure 7: Comparison of standard GP (solid line), GP with duplicate suppression (i.e., $n=1$; dotted line), and surrogate-supported GP runs (dashed lines). Indicated by the shapes of the dashed lines we investigated three different settings for the number of preselection individuals ($n=2$: circles; $n=5$: squares; $n=10$: diamonds).

As can be seen in Figure 8(d), all settings for $n$ dominate the performance of standard GP. Comparing $n=1$ with $n=2$, that is, using removal of duplicates and duplicate removal with preselection with $n=2$, both lead to about the same performance level (slightly better result of $n=2$ is not significant). Given the faster convergence, $n=2$ is a much better choice. Increasing $n$ further accelerates convergence, but performance for large budgets stagnates, in that performance of $n=10$ is significantly worse than that of $n=1$ or $n=2$. At an intermediate budget of 5,000 fitness evaluations (shown in Figures 8(a) and 8(b)) a high setting of $n$ is still beneficial, however increasing $n$ from 5 to 10 results only in a nonsignificant improvement of performance.

We explain this by the search bias introduced with the surrogate model. As long as there only is a modest influence of the model, convergence is not affected. If the model's
influence is high due to a large \( n \), errors introduced by the model can lead to premature convergence in suboptimal parts of the search space. Furthermore, given the results of Section 6.3, it seems to be more easy to differentiate between good and bad rules in the first few generations when individuals in the population are very dissimilar with a large variance of fitness values.

When optimizing with expensive fitness functions, detailed convergence lines as shown in Figure 7 will generally not be known in advance. Therefore, given a certain computational budget, a robust setting is required, which combines fast convergence with a good convergence level and low computational overhead. Given these requirements, using the phenotypic surrogate with a moderate setting of \( n = 2 \ldots 3 \) seems to be a good compromise to achieve faster convergence without sacrificing the final convergence level achievable.

### 6.5 Runtime Considerations

Using surrogate-assisted GP as presented in this paper requires additional computational time compared to standard GP. For each generation, that is, iteration of the main loop of Figure 2, this overhead consists of:

1. The time to detect duplicates and replace them with additional rules;
2. The time to build or update the surrogate model (in our case only an update of the database);
On Using Surrogates with Genetic Programming

3. Additional time to compute the phenotypic characterization, that is, in our experiments the decision vector as described in Section 4.2;

4. Time to estimate each rule’s fitness by querying the surrogate model using the rule’s phenotypic characterization;

5. Time to sort all rules by estimated fitness and copy rules with best estimated fitness to the new population.

Looking at the runtime requirements of standard GP and surrogate-assisted GP (with \( n = 2 \)) in our experiments, both running for 60 generations using 30,000 full fitness evaluations, we measured an increase of computational time of 295 s (±18 s based on a 95% confidence interval). This is an increase of less than 1.7% in total CPU time used. In our experiments, a single fitness evaluation takes about 0.6 s on average. Our measured runtime difference is therefore equivalent to about 492 (±30) fitness evaluations in total or 8.2 fitness evaluations per generation. Contrasting this with the corresponding curves in Figure 7 (dashed curve with circles) shows a clear benefit of using the surrogate for the dynamic job shop scheduling scenario used in this paper.

Even more important, this additional runtime requirement for using the surrogate does not depend on the runtime of a single simulation. When using our approach with more complex and realistic simulations taking much longer, the overhead of Surrogate-Assisted GP would soon become negligible and benefits like faster convergence to better performance levels even more important.

6.6 Influence of Population Size

This paper aims at demonstrating the benefit of integrating into GP a surrogate model based on phenotypic characterization. The relative performance of GP with and without surrogate model should be mostly independent of the GP parameter settings, as long as they are reasonable. The GP with surrogate model, however, uses an intermediate population of larger size, so it seems sensible to investigate the influence of the population size.

We therefore report in Figure 9 the results of using different population sizes in addition to the standard setting of 500 used so far in our experiments: a smaller population size of 250 and a larger setting of 1,000. Convergence curves using these three settings for standard GP are shown in Figure 9(a). As can be seen, using a population size of 250 leads to faster convergence for small computational budgets at the expense of worse mean performance for budgets larger than 7,000–8,000. A population size of 1,000 might only be beneficial for very large budgets above 20,000. Even for a budget of 30,000, however, performance differences to a setting of 500 are not statistically significant. Therefore, a population size of 500, as used for the experiments in this paper, seems to be a good compromise for the maximum budget of 30,000 fitness evaluations used.

More important for the findings of this paper is the comparison between standard GP and surrogate-assisted GP for population sizes of 250 and 1,000 as shown in Figure 9(b). For both settings there is a clear benefit of using the surrogate, resulting in faster convergence as well as convergence to better performance levels. The benefits of using the surrogate model for a population size of 500 were already analyzed in more detail in the previous sections and shown graphically in Figure 7(b). In summary, using the surrogate model appears to be beneficial independent of the population size used. For all settings investigated results with surrogate show faster convergence to better
Figure 9: Influence of population size.

6.7 Perfect Surrogate

So far, we demonstrated a clear benefit of surrogate-Assisted GP in terms of convergence speed and convergence level as compared to standard GP. Now we would like to understand how much better we might be able to do by further improving the surrogate model. To answer this question, we performed additional experiments for surrogate-assisted GP with $n = 2$. In these experiments, we replaced the calculation of performance levels ($p < .001$ based on a paired, nonparametric Wilcoxon signed rank test after 30,000 fitness evaluations).
the phenotypic characterization and surrogate evaluation with a full fitness evaluation. Of course this takes much more time, as we are performing 60,000 instead of just 30,000 fitness evaluations, but offers a lower bound, and allows us to assess the performance if we had a perfect surrogate model.

The results of these runs are shown in Figure 10. Solid and dashed curves show the performance of standard GP and surrogate-assisted GP as already shown above. Results for the hypothetical perfect surrogate are shown by the dash-dot curve. The two curves are very close, which means that the surrogate model based on a rule’s phenotypic characterization, as used in this paper, is quite effective and, at least for the parameters investigated, allows a large portion of the improvement potential to be gained over Standard GP.

We explain this by the specific setting we use for $n$. Using $n = 2$, as we do, requires the surrogate to classify individuals into the top or bottom 50% of the intermediate population (which is later on fully evaluated). The simple nearest neighbor regression model seems to be sufficient to make this distinction. Any misranking in the top 50% is later on corrected by the full fitness evaluation, errors around the 50th percentile only have a very small influence on further EA progress, as their probability of selection for poor individuals and thus the further progress of the optimization run is very small. Only large errors have a negative effect on EA performance, that is, classifying a very bad individual in the top 50% or a very good individual in the lower 50%. The higher $n$ is set, the more difficult this task becomes, however. Using a setting of, for example, $n = 10$ would require identifying the top 10% in the intermediate population, a task which requires a considerably more accurate surrogate. Therefore, a better surrogate function is probably beneficial with a higher setting for $n$.

7 Conclusion

In this paper, we presented a novel approach to improve the convergence speed of GP with expensive fitness evaluations. Introducing the concept of a phenotypic
characterization, we enable the use of surrogate models in combination with GP. The idea is to record the outcome of the rule encoded in a GP individual in a specified set of decision situations, and work in the space of phenotypic characterizations rather than genotypes to construct and evaluate the surrogate model.

We have demonstrated the usefulness of our approach at the example of evolving dispatching rules for dynamic job shop scheduling. We present a phenotypic characterization for this problem, and show how it can be used to detect (and remove) phentoypically equivalent individuals, leading to a faster evolution of better solutions.

We also show how the phenotypic characterizations can be used to successfully learn surrogate models that can be used to replace expensive fitness evaluations, further speeding up evolution. Together, our proposed approach saved up to 80% of the fitness evaluations compared to the standard GP model. The paper thus makes a contribution to allow GP to better be used for problems with computationally expensive fitness evaluations. A comparison with an alternative approach that uses a genotypic distance measure on the GP trees shows that the surrogate models built on phenotypic characterizations are much more accurate than those based on genotypic distances.

There are several ways to extend our work in the future. First, different modeling techniques can be investigated such as neural networks, support vector regression, or Gaussian process regression. Such techniques may lead to even better surrogate quality and allow higher values of \( n \) to be used effectively. It would certainly be interesting to compare their performance with the simple nearest neighbor regression used in this paper, which was already shown to achieve a large fraction of the improvement potential for \( n = 2 \) in Section 6.7.

Second, there are several ways to improve the model management and integration of the surrogate in the GP run. What training data to use and what individuals to evaluate with the expensive fitness functions are currently handled in an effective, yet simple way, certainly allowing for further refinements and improvements in future work.

Third, the idea presented in this paper should also be tested in other application domains, which would require the development of corresponding phenotypic characterizations. To what extent our approach is applicable to other optimization approaches with complex genotype-phenotype-fitness mappings besides GP, is also an interesting area for future work. A promising example for the latter would be to support the evolution of neural networks. In Branke et al. (2014), we investigated feed-forward neural networks with a single hidden layer (with weights optimized by CMA-ES; Hansen and Ostermeier, 2001) as an alternative representation to expression trees (optimized with GP) for dispatching rules. Approaches such as the one presented in Jin et al. (2002) to support CMA-ES with surrogate functions could be applied to the genotypes directly (as in Jin at al.’s paper), or using our phenotypic characterization. Using the genotype, that is, the weights of the neural network, probably has far less predictive power than using the phenotypic characterization to predict the fitness of candidate rules.

Last but not least, it would be interesting to combine the work presented in this paper with efforts to create improved GP operators incorporating semantic information, such as proposed by Uy et al. (2011), and fitness sharing incorporating semantic information (Nguyen et al., 2012). Both areas seem to be complementary to our work and can probably be combined to further improve the performance of GP in application areas with expensive fitness evaluations.
On Using Surrogates with Genetic Programming

Acknowledgments
The authors would like to thank Alberto Moraglio for the source code to compute the structural Hamming distance (SHD) as a syntactic/genotypic similarity measure between trees. This work was partially supported by the German Research Foundation (DFG) under grant SCHO 540/17-2.

References


T. Hildebrandt and J. Branke


On Using Surrogates with Genetic Programming


