European Graduate Student Workshop on Evolutionary Computation

3rd European Workshop, EvoPhD 2008
Naples, Italy, March 27, 2008
Proceedings

ISSN 1974-4145

Chairs: Cecilia Di Chio, Mario Giacobini and Jano van Hemert

Preface

Evolutionary computation involves the study of problem-solving and optimization techniques inspired by principles of evolution and genetics. As any other scientific field, its success relies on the continuity provided by new researchers joining the field to help it progress. One of the most important sources for new researchers is the next generation of PhD students that are actively studying a topic relevant to this field. It is from this main observation the idea arose of providing a platform exclusively for PhD students.

From the perspective of a PhD student, it is important to work on an exciting topic that is of interest to other researchers in the field. Contact with these researchers early in the study can help in guiding the course by focusing on what other researchers deem important. Furthermore, those contacts are important for the social network of the PhD student, which will become vital for choosing the step following the PhD study.

We would like to stress that the work presented in the proceedings is not that of regular papers. Instead it is a collection of works where each student has described the project of his/her PhD thesis, and has outlined his/her achievements and goals of the PhD study. Moreover, a description is given of the desired feedback.

This volume contains the proceedings of EvoPhD 2008, the Third European Graduate Student Workshop on Evolutionary Computation. It was held in Naples, Italy on 27 March 2008, as part of the EvoWorkshops 2008, which are part of Evo⋆: Europe's premier multi-conference and multi-workshop event on evolutionary computation and relating methodologies.

We would like to acknowledge the work performed by the Program Committee, who has made sure, with a rigorous reviewing process, that this proceedings contains scientific works of a high quality. This year, four manuscripts were accepted for publication in the proceedings and, subsequently their authors have presented their work at the workshop. Each manuscript was reviewed independently by three reviewers, which gave insightful comments on the work as presented, but also on the topic itself and the proposed work and goals.

We thank Jennifer Willies for her administration of and efforts in both the conferences and the workshops, Ivanoe De Falco, ICAR-CNR; Antonio Della Cioppa, University of Salerno; Ernesto Tarantino, ICAR-CNR, and Giuseppe Trautteur, University of Naples Federico II for making all the local arrangements.

March 2008

Cecilia Di Chio, Mario Giacobini and Jano van Hemert
Program Chairs EvoPhD 2008

Organisation

EvoPhD 2008 is organised as one of the workshops of the EvoWorkshops 2008, which are part of Evo⋆: Europe's premier multi-conference and multi-workshop annual event on evolutionary computation and relating methodologies

Organising Committee

EvoPhD Chairs:	Cecilia Di Chio University of Essex, United Kingdom
	Mario Giacobini University of Torino, Italy
	Jano van Hemert University of Edinburgh, United Kingdom
EvoWorkshops Chair:	Mario Giacobini University of Torino, Italy
Local Chair:	Ivanoe De Falco ICAR-CNR, Naples, Italy
Publicity Chair:	Anna I. Esparcia Alcázar Instituto Tecnológico de Informática, Spain

Program Committee

Ernesto Costa, Portugal
JJ Merelo, Spain
Stefano Cagnoni, Italy
Bill Langdon, United Kingdom
Angelo Cangelosi, United Kingdom
Yossi Borenstein, United Kingdom
Conor Ryan, Ireland
Leonardo Vanneschi, Italy
Alberto Moraglio, Portugal
Günter Raidl, Austria
Maurice Clerc, France
Christine Solnon, France

Table of Contents

Fitness Landscape Analysis for the Continuous Flow-shop Scheduling Problem 1
Jens Czogalla

An Analysis of Genetic Programming Operator Bias regarding the Sampling of Program Size with Potential Applications 15
Stephen Dignum

Evolving Tactical Teams for Shooter Games using Genetic Programming . 29
Darren Doherty

Optimization of the Operation of Pipeless Plants by an Evolutionary Algorithm and Simulation 43
Sabine Piana

Fitness Landscape Analysis for the Continuous Flow-shop Scheduling Problem

Jens Czogalla (Supervisor: Andreas Fink)

Helmut-Schmidt-University / UniBw Hamburg
Holstenhofweg 85, 22043 Hamburg, Germany
czogalla@hsu-hh.de

Abstract. The fitness landscape of the continuous flow-shop scheduling problem is investigated by examining the ruggedness of the landscape and the correlation between the quality of a solution and its distance to the optimal solution. The results confirm the presence of a big valley structure as known from other combinatorial optimization problems. The suitability of the landscape for search with evolutionary computation and local search methods is discussed. Additionally some distance measures for the continuous flow-shop scheduling problems are reviewed.

Key words: Evolutionary computation, Combinatorial optimization, Continuous (no wait) flow-shop scheduling, Fitness landscape analysis

1 Introduction to the Research Area

Evolutionary computation (EC) is known to perform well on a wide variety of hard optimization problems. It is receiving a lot of attention by practitioners due to its applicability to a wide range of applications and straightforward options for hybridization with problem-specific procedures. In the following we will provide a short introduction into EC and a short review of combinatorial optimization problems, especially the continuous (no-wait) flow-shop scheduling problem (CFSP) which will be the topic of this paper.

1.1 Evolutionary Computation

The term EC (or EA, evolutionary algorithms) was introduced to bring together optimization techniques simulating various aspects of evolution, namely genetic algorithms (GA), evolution strategies (ES), and evolutionary programming (EP). The common basic idea is that individuals of a population interact with one another in order to create new individuals containing information inherited from the parent solutions. Probabilistic operators such as selection, recombination, and mutation are employed to guide the evolution process, thus evolving the population towards better fitness values. An overview over EC is provided in, e.g., [1, 2]. Prominent variants of EC are GA [13, 9], particle swarm optimization (PSO) [18], and scatter search (SS) [12] which uses local search procedures as

a main feature. Current research generally shows the advantageousness of the hybridization of EC with local search in order to obtain high-quality results; see, e.g., [25]. See [15] for a detailed discussion on local search.

1.2 Combinatorial Optimization Problems

A combinatorial optimization problem (COP) consists of a set of solution elements which have to be combined in a way that certain conditions or constraints are satisfied. Combinations of the solution elements form the potential solutions of the problem and are called feasible or valid if the constraints are fulfilled. Solutions are evaluated by an objective function and the goal is to find solutions with (near) optimal objective functions values. Many (but not all) COP are computationally hard (NP-hard) and are solved by searching exponentially large search spaces [15, pages 13–16]. We concentrate on COP with permutation-based solution spaces, i.e., solutions are represented as a sequence (permutation) of the solution elements. In this paper we consider the continuous (no wait) flow-shop scheduling problem (CFSP) as posed by van Deman and Baker [30]. The CFSP consists of a set of n jobs which have to be processed in an identical order on m machines. Each machine can process exactly one job at a time. The processing time of job i on machine j is given as t_{ij} with $1 \leq i \leq n$ and $1 \leq j \leq m$.

The continuous flow-shop scheduling problem includes no-wait restrictions for the processing of each job, i.e., once the processing of a job begins, there must not be any waiting times between the processing of any consecutive tasks of this job. Continuous processing leads to a delay $d_{ik}, 1 \leq i \leq n, 1 \leq k \leq n, i \neq k$ on the first machine between the start of jobs i and k when i and k are processed directly after each other. The delay can be computed as

$$d_{ik} = \max_{1 \leq j \leq m} \left\{ \sum_{h=1}^{j} t_{ih} - \sum_{h=2}^{j} t_{k,h-1} \right\}. \tag{1}$$

The processing order of the jobs is represented as a permutation $\Pi =< \pi_1, ..., \pi_n >$ where π_i is the job processed at the i-th position of a schedule.

The objective considered is to minimize the total processing time (flow-time)

$$F(\Pi) = \sum_{i=2}^{n} (n + 1 - i)\, d_{\pi(i-1),\pi(i)} + \sum_{i=1}^{n} \sum_{j=1}^{m} t_{ij}. \tag{2}$$

The first part of Equation 2 sums the implied delays; since the delay between two jobs affects all succeeding jobs the respective delay is multiplied with the number of following jobs. The second part is the constant total sum of processing times. A review of the CFSP can be found in [8].

1.3 Important Research Questions and Related Work

Although the research on EC goes back as early as the mid-1950s there are still many areas in the field of EC that are of great interest for researchers

and practitioners. Research in this area has often opened rather then answered questions, which consequently has lead to a widening in the range of topics. Apparently the presentation of a complete overview is a nearly impossible task. Therefore we will focus in this subsection just on some important questions.

The balance between traversing large parts of the search space to find regions with good solutions (exploration/diversification) and the thorough search for elitist solutions in those regions (exploitation/intensification) is one of the fundamental issues in EC. Apparently these are partly conflicting goals. On the one side, a diverse population is needed to explore large parts of the search space. This may impede finding high quality solutions. On the other side, a more homogenous population should enable thorough exploitation of certain regions of the search space. This can result in premature convergence since a loss of diversity in the population may impede the creation of new individuals dominating their parents and thus trapping the population in a region around a local optimum. Recent approaches to control intensification and diversification are based on some kind of measurement of population diversity. In [28] acceptance criteria based on distances for the inclusion of new individuals into the population are defined. In [25] a restart mechanism is employed in case the search process is not able to produce new best solutions indicating a loss of diversity in the population. In [7] we investigate the role of local search as intensification method and its influence on population diversity.

Another important issue is the adaption of EC as a general metaheuristic to a certain problem and classes of problems. The *"no free lunch"* (NFL) theorems (see, e.g., [32]) state that if an algorithm performs well on average for one class of problems it generally performs worse on average over the remaining problems. To adapt an algorithm to a problem the structure of the problem must be considered for the choice and design of a particular algorithm [32]. Choosing a proper representation, designing new domain specific operators, and hybridizing with problem-specific heuristics and local search are ways to do so. Additionally constraint handling techniques (e.g., penalty functions or repair algorithms) or special population structures can increase the effectiveness of EC for a certain problem. See [1, 2] for a detailed overview. Related to that issue is the prediction of algorithm performance. The fitness landscape analysis [15, ch. 5], which will be discussed in more detail in Section 3, can be used. The properties of a search space landscape are examined in order to extract structural information which may be used to predict the efficiency of EC components.

2 Own Research and Study

As mentioned in the previous section controlling the balance between diversification and intensification is one of the fundamental issues in EC. So the thesis aims at gaining a better understanding of the involved mechanisms. That is, how recombination operators, selection methods, or local search (as intensification method) influence the diversity and the quality of the population and thus the search process and might be used to control it. Consequently the question

of how to measure diversity is of interest. With the results we want to develop approaches for controlling convergence behavior and thus improving the performance. Secondly, we aim for getting more insight into how the structure of a problem and the performance of EC are connected.

In [8] we investigated the critical success factors of discrete particle swarm optimization for solving COP (CFSP). The results show that the utilization of a combination of different recombination methods yields better solutions. Furthermore the hybridization with a powerful local search procedure is shown to be a requirement for obtaining high quality solutions. The fitness landscape analysis, which is topic of Section 3, will be used in Section 4 to explain the observed results. In [7] we examined the influence of different selection methods and local search procedures on population diversity. We currently continue the research on convergence behavior using the metrics reviewed in Section 3 with the goal of implementing appropriate mechanisms to control the balance between diversification and intensification and thus improving the performance of EC.

In the future we will also consider more challenging permutation-based combinatorial optimization problems, e.g., multiobjective optimization problems and problems with additional constraints. The PhD studentship is laid out for a period of three years with two years remaining.

3 Results

3.1 Methodology

The adaptation of a certain EC algorithm to a specific combinatorial optimization problem (see Subsection 1.3) can benefit from the analysis of the fitness landscape of the problem. A fitness landscape is induced by a particular operator which defines a neighborhood structure [19]. More precisely, a fitness landscape is a labeled, directed graph [16, ch. 2]. Since we employ two unary neighborhood operators for the CFSP we can simplify Jones's model [16]: The vertices of the graph correspond to solutions (i.e., permutations) of the combinatorial optimization problem (CFSP). A directed edge connecting vertice Π to Π' indicates that Π' is reachable from Π with one application of the neighborhood operator (i.e., Π is a neighbor of Π' under the considered neighborhood operator). The vertices of the graph are labeled with the corresponding fitness values, giving raise to the landscape image when thought of as heights [16, ch. 2]. Based on this definition seven *position types* for points in the search space can be defined according to the topology of their local neighborhood [15, p. 211f]:

- strict local minima (SLMIN, all neighbors have larger fitness values)
- local minima (LMIN, no neighbor has a smaller fitness value)
- interior plateau (IPLAT, all neighbors have the same fitness values)
- ledge (LEDGE, all types of neighbors)
- slope (SLOPE, no neighbor has the same fitness value)
- local maxima (LMAX, no neighbor has a larger fitness value)
- strict local maxima (SLMAX, all neighbors have smaller fitness values)

Every search space position can be assigned to exactly one position type. For small instance sizes the exact *position type distribution* can be determined by total enumeration (see Subsection 3.3 for results). Sampling methods have to be applied for larger instance sizes. For a detailed discussion we refer to [15, p. 213].

Several global features of fitness landscapes influence the performance of heuristic optimization algorithms [20]. In this paper we will concentrate on statistical methods for measuring the landscape ruggedness and for analyzing the distribution of local optima in the search space.

The *ruggedness* of a landscape is a measure for the correlation of fitness values of neighboring points in the search space and for the number of local optima. It can be used to predict the performance of local search procedures incorporated in EC [3]. A landscape with a high density of distinct local optima is supposed to be very rugged and hence hard to tackle by local search methods. On the other side a smooth landscape ought to have fewer local optima enabling a local search procedure to perform a larger number of iterations before finding a local optimum, thus exploring larger parts of the search space [15, p. 226f]. As measure for the ruggedness of a fitness landscape the *empirical autocorrelation function* can be utilized. It provides statistical information about the correlation between two points that are s steps apart (under a certain operator) and is calculated as

$$r(s) = \frac{1}{\sigma^2(f)(m-s)} \sum_{t=1}^{m-s} (f(x_t) - \overline{f})(f(x_{t+s}) - \overline{f}) \tag{3}$$

where f is a series of fitness values $(f_1, ..., f_m)$ generated by means of a random walk of m steps starting at a randomly selected position in the search space. The variance of the fitness values is given by

$$\sigma^2 = \frac{1}{m} \sum_{k=1}^{m} (f_k - \overline{f})^2 \tag{4}$$

with $\overline{f}$ as the average of the sequence of fitness values [15, p. 228]. The most important correlation is the *first-order correlation* $r(1)$ (or nearest-neighbor correlation of the landscape) which describes the dependence between neighboring positions. Values of $r(1)$ that are close to one indicate that neighboring positions tend to have a similar level of fitness values, corresponding to a smooth landscape. Values of $r(1)$ close to zero suggest that there is no correlation of the fitness values between neighboring positions, thus implicating a very rugged landscape. Based on the empirical autocorrelation function, the *correlation length* l of the landscape is defined as

$$l = -\frac{1}{ln(|r(1)|)} \qquad \text{for } r(1) \neq 0 \quad \text{and} \quad r(1) \neq 1. \tag{5}$$

The correlation length summarizes the ruggedness of a landscape. The larger the correlation length, the smoother the search landscape. Since the correlation length typically depends on instance size it is often normalized to allow the

comparison of correlation lengths for different instance sizes and for different neighborhood structures [15, p. 229]:

$$l' = \frac{l}{diam} \tag{6}$$

with *diam* as the *diameter* of the landscape graph. It is defined as the maximal distance between any two vertices Π and Π', i.e., the number of edges that have to be traversed to reach Π' from Π.

In [16] the *fitness distance correlation* (FDC) as a measure of problem difficulty for EC algorithms is proposed. FDC is the correlation between the quality of a solution and its distance to an optimal solution. In case of two or more optimal solutions the distance to the closest optimal solution may be considered. Since FDC requires the definition of a distance measure we will present some metrics for the CFSP later in this section.

The FDC states how closely fitness and distance to an optimal solution are related. If fitness increases when the distance to the optimum becomes smaller, the search is expected to be relatively easy for selection-based algorithms, since the evolution of the population is guided to a global optimum via solutions with increasing fitness [20]. The FDC coefficient can be computed based on a sample of m solutions f and the corresponding distances d to the optimal solution. The FDC coefficient is defined as [15, p. 222f]

$$\varrho(f, d) = \frac{cov(f, d)}{\sigma(f)\sigma(d)} \tag{7}$$

where

$$cov(f, d) = \frac{1}{m-1} \sum_{i=1}^{m} (f_i - \overline{f})(d_i - \overline{d}) \tag{8}$$

$$\sigma(f) = \sqrt{\frac{1}{m-1} \sum_{i=1}^{m} (f_i - \overline{f})^2}, \qquad \sigma(d) = \sqrt{\frac{1}{m-1} \sum_{i=1}^{m} (d_i - \overline{d})^2}. \tag{9}$$

A high value of ϱ indicates that fitness and distance to the optimum are related. In [17] a classification of problem difficulty based on FDC coefficients is suggested. A problem with $\varrho \geq 0.15$ (in [17] maximization problems were investigated) is called *straightforward* and *should* be suitable for GA and more general for EC.

The FDC coefficient can be estimated using randomly generated samples [16, ch. 5]. It is more interesting to focus the analysis towards locally optimal solutions since local search procedures (as important components of EC) are biased towards sampling good solutions. Fitness distance scatter plots can support the evaluation of the FDC coefficient by avoiding misinterpretations.

Based on FDC analysis a *big valley* structure was reported for some optimization problems, e.g., the traveling salesman problem [4], the graph bisection problem [4], the permutation flowshop sequencing problem [23], and the quadratic assignment problem [20]. The *big valley* structure means that local

optima tend to be relatively close to each other and to the global optimum. High FDC coefficients are an indicator for the presence of the big valley structure. Further evidence for a big valley structure can be obtained from results on the correlation between the solution quality and the average distance between one element and any other element of a given set of locally optimal solutions [4]. The presence of a big valley structure should support the performance of recombination in population-based search. Usually the initial population of solutions is uniformly distributed over the complete search space. Recombination should focus the search in a region with good solutions. In a big valley structure recombination can potentially drive the search towards the optimal solution.

In the following we review some distances for the CFSP. We consider two different notions of distance. Derived from the interpretation of the permutation representation where either the adjacency relation among the elements of the permutation, or the relative order of the elements, or the absolute position of the elements may be of relevance we review the *adjacency distance*, the *precedence distance*, and the *absolute position distance*. Related to the absolute position distance is the *deviation distance* which is a measure for positional deviation of two permutations. The distance between two points (solutions) can also be defined as the minimal number of elementary move operators which have to be applied to transform one permutation into the other permutation. Based on two neighborhood structures known to be suitable for the CFSP [11] we review the *swap distance* and the *shift distance*. Additionally we consider the *edit distance* which is based on edit operations.

The unidirectional version of the *adjacency distance* (adjacency based distance in [26]) is defined depending on the number of times a pair of jobs i, j is adjacent in both Π and Π':

$$d(\Pi,\Pi') = n-1-\sum_{i=1}^{n-1} y_i \qquad \text{with } y_i = \begin{cases} 1 & \text{for } \pi_i = \pi'_j \text{ and } \pi_{i+1} = \pi'_{j+1} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

with the maximum adjacency distance $d_{adj,max} = n-1$.

The *precedence distance* is defined depending on the number of times n_{pre} some job j is preceded by job i in both Π and Π' [23]:

$$d_{pre}(\Pi,\Pi') = \frac{n(n-1)}{2} - n_{pre} \tag{11}$$

with the maximum precedence distance $d_{pre,max} = n(n-1)/2$.

The *absolute position distance* (exact match distance in [24]) is defined depending on the number of exact positional matches of jobs [24]:

$$d_{abs}(\Pi,\Pi') = n-\sum_{k=1}^{n} y_i \qquad \text{with } y_i = \begin{cases} 1 & \text{if } \pi_i = \pi'_i \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

with the maximum absolute distance $d_{abs,max} = n$.

The *deviation distance* (position-based metric in [23]) requires the definition of the inverse permutation Σ of sequence Π, where the position of job π_i is given

by $\sigma_{\pi_i} = i$. The deviation distance is the amount of positional deviation and can be calculated as [23]

$$d_{dev}(\Pi, \Pi') = \sum_{j=1}^{n} |\sigma_j - \sigma'_j| \tag{13}$$

with the maximum deviation distance [24]

$$d_{dev,max} = \begin{cases} n^2/2 & \text{if } n \text{ is even} \\ (n^2-1)/2 & \text{if } n \text{ is odd.} \end{cases} \tag{14}$$

The *swap distance* is based on the swap (or interchange) move which exchanges a pair of jobs π_i and π_j with $i \neq j$. The calculation of the exact swap distance between two permutations is nontrivial. In order to reduce computational complexity a path in the swap neighborhood may be calculated by position-wise comparison of the two parent permutations Π and Π'. If a job in Π is not in the same position as in Π' it is swapped to the correct position and the move is stored. The number of stored moves corresponds to the approximated swap-distance $d_{swap}(\Pi, \Pi')$ with the maximum swap distance $d_{swap,max} = n-1$.

The *shift distance* is based on the shift (or insertion) move which removes the job π_i and inserts it behind the job π_j with $i \neq j$ and $i \neq j+1$. The computation of the shortest shift distance (or permutation edit distance in [6]) is related to the longest common subsequence of the two permutations Π and Π'. Let $LCS(\Pi, \Pi')$ be the length of the longest common subsequence of Π and Π', the shift distance can be calculated as [6]

$$d_{shift}(\Pi, \Pi') = n - LCS(\Pi, \Pi') \tag{15}$$

with the maximum shift distance $d_{shift,max} = n - 1$.

The *edit distance* between two permutations Π and Π' is the minimal number of edit operations required to transform permutation Π into Π'. The elementary edit operations are *substitution*, *insertion*, and *deletion*. A substitution changes an element into another one, a deletion removes an element, and an insertion inserts an element [27]. Under the assumption that the cost of an insertion or a deletion is weighted as 1 and the cost of a substitution is 2 then the edit distance can be calculated by [31]

$$d_{edit}(\Pi, \Pi') = 2(n - LCS(\Pi, \Pi')) \tag{16}$$

with the maximum edit distance $d_{edit,max} = 2(n-1)$.

Normalizing the distances shows that under the assumed costs edit distance and shift distance are equal.

3.2 Computational Experiments

For the fitness landscape analysis flow-shop scheduling problem instances were taken from the OR-Library[1] and treated as CFSP instances, namely the problem instances provided by Taillard (TA) [29], Heller (HEL) [14], Carlier (CAR)

[1] `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/flowshopinfo.html`

[5], and Reeves (REC) [22]. The latter were taken into account since they contain small instances which are solvable by exhaustive enumeration. Additionally two 10x5 instances were created using Taillard's instance generator (with seeds 873654221 and 379008056).

The 3-index formulation of Picard and Queyranne [21] was used to compute optimal solutions for the instances with up to 50 jobs (see Table 1). For the larger instances the FDC analysis was based on the best solutions presented in [8] (TA) and the best values found during the experiments (HEL, CAR, REC) respectively. The data for the FDC analysis was obtained by performing multiple runs of iterative improvement algorithms with the swap and the shift neighborhood as described in [15, p. 222f]. The number of distinct local optima found is given in Table 1. Since the samples apparently do not consist of independent random samples a standard hypothesis test is not applicable. A randomization test [10] was used to determine the significance of the found FDC coefficients (as in [23]).

Table 1. Best solutions, number of local optima, normalized correlation length, and first order correlation for the investigated instances. In case of a provably optimal solution the entry is marked with an asterisk.

instance	n x m	best solution	local optima		l'		$r(1)$	
			swap	shift	swap	shift	swap	shift
CAR7	7 x 7	36,534*	5	4	0.41	0.46	0.67	0.70
CAR6	8 x 9	52,946*	25	4	0.35	0.42	0.67	0.71
CAR8	8 x 8	52,703*	25	4	0.33	0.38	0.65	0.69
CAR5	10 x 6	58,445*	155	22	0.40	0.46	0.76	0.78
873654221	10 x 5	5,307*	163	11	0.34	0.43	0.72	0.77
379008056	10 x 5	5,671*	339	11	0.32	0.41	0.71	0.76
CAR1	11 x 5	52,353*	599	13	0.36	0.40	0.76	0.78
HEL2[a]	20 x 10	2,069*	2,493	2,359	0.39	0.37	0.87	0.87
REC01	20 x 5	17,187*	2,492	1,487	0.30	0.42	0.84	0.88
REC03	20 x 5	14,632*	2,467	961	0.31	0.43	0.84	0.86
REC07	20 x 10	24,598*	2,483	1,329	0.33	0.41	0.85	0.88
TA001	20 x 5	15,674*	2,443	1,271	0.34	0.43	0.86	0.86
TA011	20 x 10	25,205*	2,496	1,921	0.29	0.39	0.84	0.87
REC031	50 x 10	112,698*	2,000	1,814	0.29	0.33	0.93	0.94
TA031	50 x 5	75,668*	2,000	2,000	0.32	0.37	0.94	0.95
TA061	100 x 5	303,273	2,000	2,000	0.32	0.34	0.97	0.97
TA091	200 x 10	1,488,029	2,000	2,000	0.21	0.26	0.98	0.98

[a] For the problem instance HEL2 two global optima were found.

3.3 Results of the Landscape Analysis

To determine the ruggedness of the fitness landscape the distribution of the position types for the seven smallest problem instances was determined by enumerating all possible solutions. The results are presented in Table 2. The values

present the percentage of the total number of search space positions and the numbers in brackets stand for the absolute count. The first line for each instance represents the values w.r.t. the swap neighborhood and the second one for the shift neighborhood. One can see that the number of (strict) local optima (w.r.t. both swap and shift neighborhood) is very small and that for no problem instance a search space position of type *IPLAT* was found. That should make both fitness landscapes favorable for local search procedures. One can expect a relatively large number of iterations before a local optimum is detected. The absence of interior plateaus prevents local search getting trapped in regions where the search for exits may impede the search process.

Table 2. Position type distribution for the smaller instances.

instance	SLMIN	LMIN	IPLAT	SLOPE	LEDGE	LMAX	SLMAX
CAR7	0.10 (5)	0.00 (-)	0.00 (-)	99.48	0.20	0.00 (-)	0.22 (11)
	0.08 (4)	0.00 (-)	0.00 (-)	99.03	0.83	0.00 (-)	0.06 (3)
CAR6	0.06 (25)	0.00 (-)	0.00 (-)	99.48	0.40	0.00 (-)	0.06 (26)
	0.01 (4)	0.00 (-)	0.00 (-)	99.22	0.73	0.00 (1)	0.04 (16)
CAR8	0.06 (25)	0.00 (-)	0.00 (-)	99.24	0.59	0.00 (1)	0.10 (40)
	0.01 (4)	0.00 (-)	0.00 (-)	98.89	1.06	0.00 (-)	0.04 (17)
CAR5	0.00 (155)	0.00 (-)	0.00 (-)	99.25	0.74	0.00 (-)	0.00 (92)
	0.00 (22)	0.00 (-)	0.00 (-)	98.72	1.27	0.00 (2)	0.00 (154)
873654221	0.00 (157)	0.00 (6)	0.00 (-)	90.69	9.30	0.00 (22)	0.00 (126)
	0.00 (11)	0.00 (-)	0.00 (-)	87.11	12.89	0.00 (5)	0.00 (59)
379008056	0.01 (311)	0.00 (28)	0.00 (-)	91.73	8.26	0.00 (12)	0.00 (111)
	0.00 (9)	0.00 (2)	0.00 (-)	86.66	13.34	0.00 (1)	0.00 (87)
CAR1	0.00 (595)	0.00 (4)	0.00 (-)	98.42	1.57	0.00 (4)	0.00 (576)
	0.00 (13)	0.00 (-)	0.00 (-)	98.12	1.88	0.00 (-)	0.00 (63)

The evaluation of the first order correlation (Table 1) suggests a smooth landscape and amplifies the idea that the landscape structure of the CFSP is favorable for local search procedures. The comparison of the normalized correlation lengths indicates that the shift neighborhood seems to be favorable for local search. The reason for this may be the larger neighborhood size (compared to the swap operator) leading to a smaller diameter and, as shown with the position type distribution (Table 1), fewer number of local optima.

The results of the FDC are presented in Table 3. The first row for each instance shows the FDC coefficients based on local optima in the swap neighborhood. The second row stands for the landscape derived from the shift operator. The first value of each entry corresponds to the FDC coefficient calculated for the distance to the best solution, the second to the average distance to all other local optima. In case a coefficient is not significant at the 0.01 level (according to a randomization test) the entry is marked with an asterisk.

For the examined problem instances the FDC coefficients indicate that, according to [17], the fitness landscapes induced by the swap and the shift operator

Table 3. Fitness distance correlation coefficients for the investigated instances. The first value corresponds to the distance to the best solution, the second to the average distance to all other global optima. The first line for each instance represents the swap neighborhood, the second one the shift neighborhood. In case a coefficient is not significant at the 0.01 level the entry is marked with an asterisk.

instance	FDC coefficient for distance metrics					
	adjacency	precedence	absolute	deviation	edit	swap
CAR7	0.517*/0.187*	0.824*/0.512*	0.817*/0.763*	0.907*/0.640*	0.735*/0.495*	0.864*/0.564*
	0.989/-0.542*	0.897*/0.938	0.856*/0.757	0.856*/0.953	0.986/0.308*	0.779*/0.757
CAR6	0.509/0.302*	0.714/0.812	0.509/0.494	0.606/0.704	0.805/0.846	0.316*/0.360*
	0.563*/-0.365*	0.951/0.220*	0.906/0.358*	0.904/0.248*	0.956/0.038*	0.920/-0.082*
CAR8	0.281*/0.372*	0.615/0.728	0.643/0.712	0.669/0.729	0.629/0.818	0.541/0.636
	0.950*/-0.108*	0.992/0.293*	0.948/0.956*	0.997/0.753*	0.999/-0.010*	0.914/0.867
CAR5	0.523/0.633	0.721/0.674	0.505/0.517	0.669/0.638	0.777/0.816	0.440/0.430
	0.688/0.603	0.861/0.508	0.485/-0.026*	0.797/0.426*	0.868/0.639	0.487/-0.116*
873654221	0.241/0.017*	0.675/0.796	0.385/0.672	0.625/0.790	0.603/0.765	0.423/0.697
	0.484*/0.649*	0.364*/0.564*	0.202*/0.293*	0.304*/0.534*	0.251*/0.633*	0.019*/0.327*
379008057	0.293/0.199	0.565/0.385	0.417/0.556	0.516/0.368	0.604/0.588	0.294/0.459
	0.535*/0.050*	0.901/0.376*	0.889/0.304*	0.877/0.087*	0.822/0.091*	0.860/0.627
CAR1	0.382/0.195	0.608/0.368	0.545/0.320	0.635/0.428	0.582/0.550	0.499/0.226
	0.612/0.578*	0.688/0.450*	0.498/0.468*	0.680/0.421*	0.492*/0.502*	0.501*/0.405*
HEL2	0.383/0.613	0.716/0.760	0.577/0.844	0.667/0.758	0.565/0.766	0.551/0.836
	0.262/0.313	0.401/0.568	0.373/0.724	0.396/0.631	0.488/0.687	0.332/0.705
REC01	0.316/0.574	0.643/0.713	0.352/0.382	0.606/0.688	0.592/0.742	0.298/0.424
	0.283/0.476	0.470/0.568	0.288/0.496	0.469/0.545	0.443/0.627	0.273/0.502
REC03	0.361/0.624	0.651/0.740	0.415/0.598	0.632/0.737	0.560/0.844	0.365/0.614
	0.259/0.380	0.616/0.666	0.510/0.589	0.629/0.660	0.506/0.746	0.479/0.592
REC07	0.504/0.663	0.681/0.680	0.385/0.672	0.638/0.672	0.679/0.836	0.387/0.685
	0.392/0.527	0.567/0.559	0.452/0.585	0.562/0.566	0.620/0.722	0.412/0.582
TA001	0.532/0.767	0.665/0.750	0.525/0.729	0.650/0.802	0.605/0.825	0.488/0.748
	0.459/0.660	0.453/0.602	0.569/0.795	0.498/0.649	0.519/0.781	0.533/0.795
TA011	0.327/0.574	0.332/0.498	0.243/0.524	0.321/0.479	0.329/0.671	0.223/0.528
	0.404/0.530	0.455/0.590	0.315/0.424	0.426/0.562	0.456/0.701	0.275/0.419
REC031	0.427/0.740	0.490/0.642	0.171/0.209	0.482/0.609	0.382/0.757	0.166/0.265
	0.335/0.528	0.451/0.560	0.222/0.399	0.425/0.537	0.434/0.540	0.217/0.429
TA031	0.701/0.869	0.677/0.794	0.407/0.660	0.653/0.787	0.637/0.863	0.403/0.701
	0.596/0.751	0.505/0.638	0.326/0.486	0.470/0.626	0.540/0.703	0.326/0.521
TA061	0.460/0.691	0.488/0.541	0.161/0.275	0.465/0.533	0.341/0.692	0.155/0.341
	0.485/0.745	0.529/0.554	0.159/0.279	0.507/0.542	0.382/0.706	0.166/0.337
TA091	0.495/0.681	0.543/0.527	0.222/0.274	0.521/0.526	0.420/0.713	0.231/0.310
	0.510/0.644	0.479/0.510	0.244/0.194	0.457/0.508	0.467/0.576	0.235/0.225

are suitable for EC algorithms and a big valley structure may be present. However, the differences between the coefficient values for different distance measures are noticeable. The question which metric is a good distance measure for the CFSP (or what properties have good solutions in common) is not that clear from the obtained results. It seems that the precedence distance may be best suited for the CFSP because it produces high correlations for all investigated instances. Additionally the average distances between local optima are quite small. In Figure 1 the scatter plot for the 20 jobs instance TA001 is exemplarily shown. The other examined instances show similar scatter plots. As distance measure the precedence distance was used for the reasons described. The distance to the best solution and the average distance to all other local optima respectively is plotted against the percentage relative deviation to the best solution. The evaluation of the scatter plots confirms the assumption of the existence of the big valley structure for the examined problem instances. The generated local op-

tima seem to be clustered around the global optimum (or best solution) in a small region of the search space. However, given the sampling methods used the samples might be biased and thus the results imprecise. The presence of the big valley structure should make the CFSP solvable for EC. Maybe recombination methods focusing on the relative order of the jobs are favorable. Local search (if not used as greedy search) has to be adapted in a way that other local optima in the cluster are reachable without straying too far away from it. This can be done by means of perturbing solutions to generate new starting points for the search or the use of different neighborhood structures.

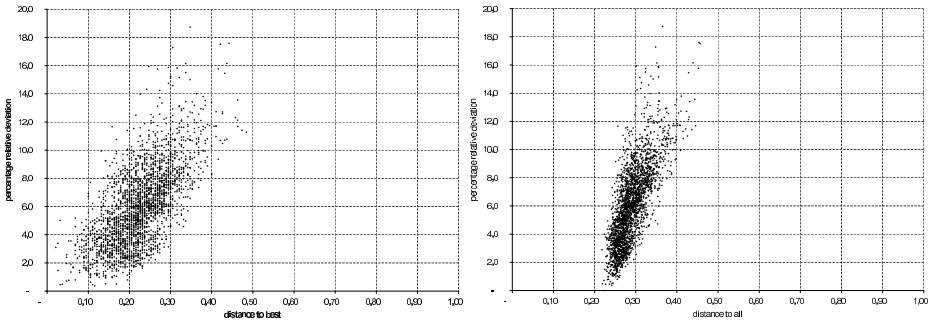

Fig. 1. Analysis of 2,443 distinct local minima w.r.t. the swap neighborhood for the 20 jobs instance TA001. The percentage relative deviation to the optimal objective function value is plotted against (left) precedence distance to the optimum solution and (right) the average precedence distance to all other local optima.

4 Achievements

The results from the fitness landscape analysis indicate that EC should be able to detect good solutions for the CFSP. Furthermore the fitness landscape of the CFSP should be workable for local search procedures. Hybridization with those domain specific improvement methods is advisable. Even if there is no superior distance measure (or measure for similarity) for all examined problem instances the relative order of the jobs might be regarded as the most important property of good solutions. This feature might be exploited by recombination operators specifically designed for the CFSP.

The obtained results help to explain the findings in [8] where we investigated the critical success factors of discrete particle swarm optimization for solving the CFSP. The combination of different crossover operators turned out to be superior compared to the application of a single one. Since the results were obtained for a whole set of problem instances the exploitation of different properties of good solutions may be responsible for the improvement. However, especially for the larger instances, results reported in the literature could not be obtained

without local search. The hybridization with local search procedures enabled the algorithm to find high quality solutions and even improve some of the best results found in the literature. Closer examination of the algorithm revealed that high quality solutions were mainly attributed to local search. In [11] tabu search and simulated annealing were used to solve the CFSP. Despite limited computational resources high quality solutions could be obtained emphasizing the adequacy of the CFSP fitness landscape for local search procedures.

5 Feedback

With the submission of this work to EvoPhD we hope to get feedback if the proposed thesis outline is suited for a good thesis on the topic of EC and COP. Furthermore we are looking forward for comments on what EC issues should additionally be included and what problem classes would make a nice proceeding of the work done so far. Generally every comment is welcome, also critical ones, even if that means "back to the drawing board".

References

1. T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators*. Taylor & Francis, New York, NY, 2000.
2. T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 2: Advanced Algorithms and Operators*. Institute of Physics Publishing, Bristol, 2000.
3. C. Bierwirth, D. C. Mattfeld, and J.-P. Watson. Landscape regularity amd random walks for the job-shop scheduling problem. In J. Gottlieb and G. R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization, 4th European Conference, EvoCOP 2004*, volume 3004 of *Lecture Notes in Computer Science*, pages 21–30. Springer, Berlin, 2004.
4. K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16:101–113, 1994.
5. J. Carlier. Ordonnancements a constraints disjonctives. *R.A.I.R.O. Recherche operationelle/Operations Research*, 2:333–351, 1978.
6. G. Cormode. *Sequence Distance Embeddings*. PhD thesis, The University of Warwick, 2003.
7. J. Czogalla and A. Fink. Evolutionary computation for the continuous flow-shop scheduling problem. In M. J. Geiger and W. Habenicht, editors, *Proceedings of EU/ME 2007: Metaheuristics in the Service Industry*, pages 37–45, 2007.
8. J. Czogalla and A. Fink. On the effectiveness of particle swarm optimization and variable neighborhood descent for the continuous flow-shop scheduling problem. In F. Xhafa and A. Abraham, editors, *Metaheuristic for Scheduling in Industrial and Manufacturing Applications*, Studies in Computational Intelligence. Springer, Berlin, 2008, to appear.
9. L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, NY, 1991.
10. E. S. Edgington. *Randomization Tests*, volume 77 of *STATISTICS: Textbooks and Monographs*. Marcel Dekker, New York, NY, second edition, 1987.

11. A. Fink and S. Voß. Solving the continuous flow-shop scheduling problem by metaheuristics. *European Journal of Operational Research*, 151:400–414, 2003.
12. F. Glover, M. Laguna, and R. Marti. Scatter search. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computing: Theory and Applications*, pages 519–538. Springer, Berlin, 2003.
13. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
14. J. Heller. Some numerical experiments for an MxJ flow shop and its decision-theoretical aspects. *Operations Research*, 8:178–184, 1960.
15. H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Elsevier, Amsterdam, 2005.
16. T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, Albuquerque, New Mexico, 1995.
17. T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, 1995.
18. J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.
19. P. Merz. *Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies*. PhD thesis, Universität-Gesamthochschule Siegen, 2000.
20. P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, 2000.
21. J.-C. Picard and M. Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, 26:86–110, 1978.
22. C. R. Reeves. A genetic algorithm for flowshop sequencing. *Computers and Operations Research*, 22(1):5–13, 1995.
23. C. R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490, 1999.
24. S. Ronald. More distance functions for order-based encodings. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 558–563, 1998.
25. R. Ruiz, C. Maroto, and J. Alcaraz. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34(5):461–476, 2006.
26. T. Schiavinotto and T. Stützle. A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34(10):3143–3153, 2007.
27. K. Sörensen. Distance measures based on the edit distance for permutation-type representations. *Journal of Heuristics*, 13:35–47, 2007.
28. K. Sörensen and M. Sevaux. MA|PM: Memetic algorithms with population managment. *Computers and Operations Research*, 33(5):1214–1225, 2006.
29. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.
30. J. M. van Deman and K. R. Baker. Minimizing mean flowtime in the flow shop with no intermediate queues. *AIIE Transactions*, 6:28–34, 1974.
31. R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, 1974.
32. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. In *IEEE Transactions on Evolutionary Computation*, volume 1, pages 67–82, 1997.

An Analysis of Genetic Programming Operator Bias regarding the Sampling of Program Size with Potential Applications

Stephen Dignum

University of Essex
sandig@essex.ac.uk
dces.essex.ac.uk/research/NEC

Abstract. This paper describes research into program size bias present within Genetic Programming operators notably crossover, mutation and population initialisation. An understanding of this bias in conjunction with program space theory allows us to make informed choices during experimental design. It also allows us to explain certain effects seen during GP runs such as program growth, the sampling of smaller programs, and the effects of applying length limits. A promising aspect of the research has been in the creation of new operators that can limit resampling effects and also to provide user defined bias which can be used to determine and exploit program search space knowledge. This research has also led to the creation of a new bloat theory: Crossover-Bias.

1 Introduction of the Research Area

Program size is important to the efficiency of Genetic Programming (GP). Too large a program will produce an inefficient solution and one that is costly to quantify in terms of fitness, whilst too small a program will not produce a solution at all. The aim of this project is to analyse in-built operator bias regarding the sampling of program size during a GP run and to suggest methods of improving the GP process in light of this knowledge.

With the advent of a greater understanding of program search spaces, primarily, we now know that the functionality of programs reaches a limit as program length increases [1–3] acquiring knowledge of how GP operators sample program length classes has become more and more urgent. Ideally, we would like to sample the length class where the smallest optimal programs can be found. Unfortunately, in general: a) one does not know where solutions (let alone most compact ones) are, and, b) genetic operators present specific (length) biases which are often unknown or only partially known and, therefore, are difficult to direct and control. A characterisation of operator bias, therefore, is needed in understanding how GP will sample the search space in the first instance before technically sound problem specific modifications can be made.

Prior to this project, research has been conducted in a number of related areas. For example, first, in the discovery of sub-tree swapping crossover length distributions for variable length strings in linear GP [4]. Second, the authors also provided another significant result showing that program length will remain constant with

repeated application of sub-tree swapping crossover on a flat fitness landscape in the absence of genetic drift [5]. And finally, but not limited to, expected program sizes for traditional GP initialisation methods have been determined in [6]. Research has, however, been limited with little attempt to consolidate findings and conclusions.

2 Research and Study

2.1 Goals

The project aims to analyse the effect of GP operators on program size, initially in isolation and then in conjunction with selection and other operators, in order to answer the following questions:

Do existing GP operators, notably population initialisation, crossover and mutation have an inbuilt bias to sample certain program sizes? If this is so how can we model this bias mathematically so that we can explain the consequences of their repeated application and to understand the combined effect of their use? A mathematical model will also allow us to determine which parameters for these operators are important with regards to length bias.

We can then ask how does size bias affect the exploration of the GP search space? As size bias will cause a GP run to sample certain areas of the search space more extensively than others, we need to determine if this is beneficial or detrimental. We can then alter existing, or devise new operators to create, direct, increase or reduce this bias.

We can use the knowledge gained from this work to understand typical GP effects such as code growth (bloat), shrinkage, convergence, and program resampling.

Finally and most importantly, this research can be used in conjunction with program space theory [1–3] to improve GP search. This can take many forms from bloat controls, direct biasing of program search space exploration to the simple selection of existing operators and their parameters.

2.2 Current Status

In [7] strong theoretical and experimental evidence was provided that using a flat fitness landscape standard sub-tree swapping crossover with uniform selection of crossover points pushes a population of a-ary[1] GP trees towards a limiting distribution of tree sizes. i.e. with repeated application the length distribution of trees will converge to a fixed point. The distribution was shown to be a Lagrange distribution of the second kind. This is an important result as it shows how, when presented with individuals from the mating pool during a normal GP run with selection, crossover will bias it's exploration of the search space.

The model was generalised in [9] to enable mixed arity tree internal node distributions to be predicted. Again strong experimental evidence was provided to support the model. In both the a-ary and mixed arity tree models there was shown

[1] GP systems with functions of a certain arity only e.g. 2-ary trees for boolean type problems such as 4-bit Even Parity [8].

to be a strong bias for this form of sub-tree swapping crossover to sample exponentially smaller programs. This bias is most acute when a population is initialised with a small mean program size. The work was also extended in this paper to sub-tree swapping crossover with 90% internal node selection, 10% internal node selection policy, commonly used by GP practitioners. This again showed a strong bias towards the sampling of smaller programs.

As this form of crossover likes to sample smaller programs we chose to initialise a number of standard GP problems with a Lagrange distribution. This was easily achieved by turning off selection for a number initial generations. For many problems it was found that 'Lagrange-like' initialisation induced bloat from the earliest generations i.e. as soon as we turned selection back on. This was felt at first to be surprising as bloat is normally associated with later generations of a run when effectively the population starts stagnating. However, when we analysed the problems that exhibited these properties (small sizes correlate with relatively low fitness values) and those that don't (small sizes can achieve initially relatively high fitness values) it led us to a new bloat theory which we have called 'Crossover-Bias' [9].

More recently we have extended our models once more to produce exact length models that can be used by practitioners (previously these had been expressed in terms of internal nodes). This work is described in section 3.2.

Crossover-bias has been used to explain the effects of length limits on GP experimentation i.e. an accelerated push towards a length limit and the increased sampling of the very smallest classes as ever tighter length limits are imposed. A new bloat control technique called 'Sampling Parsimony' has also been suggested in light of this research. This has been used to show we can control program growth by applying penalties to resampling after or before certain boundaries are reached.

We have looked at how to directly manipulate the sampling, by length, of programs by GP during an experimental run. To do this we have created a technique called 'Operator Equalisation', that allows existing GP implementations, with only slight modification, to sample pre-determined length distributions. Results show that we can automatically identify potentially optimal solution length classes quickly using small samples and that, for particular classes of problems, simple length biases can significantly improve the best fitness found during a GP run.

Both Sampling Parsimony and Operator Equalisation papers have been accepted for presentation at the EuroGP 2008 conference.

2.3 Future Planning

Current research has concentrated on arguably the defining genetic operator of standard GP [10], sub-tree swapping crossover. Although impressive results have been achieved, a number of questions remain that can be tackled in the time available. Firstly, how can we extend our models to other forms of crossover? Secondly, how can this work be combined and extended to program initialisation and mutation? As the first part of our method (section 3.1) is to look at genetic operators apart from selection, this should allow us to understand how GP handles neutrality in that when areas of neutrality are reached search will become strongly influenced by the biases we have identified. Finally, this work can provide us with

a number of answers regarding the lack of convergence of GP populations to identical individuals as seen in Genetic Algorithms. As fitness converges crossover will continue to distribute individuals according to a limiting length distribution. How GP crossover searches within each length class has also yet to be analysed.

It is intended that over the next six months these questions will be tackled with the intention of starting production of the thesis after that point. Creation of journal articles and conference papers summarising and extending current research is also proposed during that time.

After submitting the thesis I would ideally like to continue my research into Genetic Programming and extend this to other forms of program space search. I have also begun the first module of a higher education teaching qualification so that I have improved chances of gaining a lectureship at a university.

2.4 Study

I am in the third year of my PhD and intend to complete within this study year or very soon after.

There is a study programme outlined for research students defined by the University of Essex. This programme is broken into Initial, Middle and Final periods, each has a number of tasks and written progress reports associated with it. Typical tasks include the production of an outline proposal, a full proposal, critical literature survey etc.

Progress is monitored by a supervisory board that sits every six months. This consists of a supervisory board chair, the PhD supervisor, and one other member of academic staff. Board members are chosen on the basis of their competence to comment on the empirical or thematic substance of a student's work. Research progress reports need to be submitted to the board and a presentation of current work and overall progress is given at every meeting.

3 Results

3.1 Methodology

The methodology employed is as follows:

First, existing research is examined. There are a number of theoretical and empirical papers that have looked at aspects of this problem but within a restricted framework. For example, initial crossover bias work looked at Linear-GP only; I have, therefore, been able to extend this work to examine tree structures.

The next aim is to extend the theory preferably using a mathematical model. This can then be used for simulation using a tool such as MatLab, or as the basis of a highly simplified computer program. For example, a test suite has been created that repeatedly applies crossover and/or mutation to a population made up of generic GP nodes that are problem independent. This also allows alteration of population initialisation and crossover/mutation parameters. In addition, a number of statistical tools have been created that are suited to the analysis of program length.

Once strong evidence of the model has been obtained in isolation, a number of standard GP problems are set up using the Java Evolutionary toolkit ECJ [11].

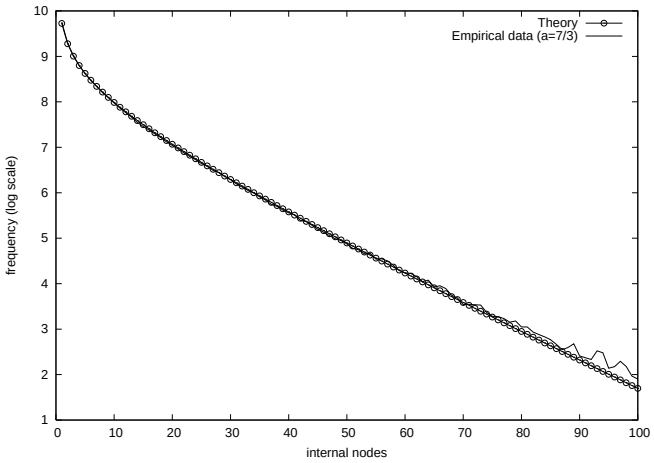

Fig. 1. Comparison between empirical and theoretical program size distributions for trees made up of a mix of 2, 2, and 3 arity functions ($\bar{a}$=7/3) initialised with FULL method (depth = 3, initial mean size $\mu_0 = 21.48$, mean size after 500 generations $\mu_{500} = 23.57$)

The effect of parameter variation, in conjunction with the application of selection, is examined in terms of eventual length distributions and also the effect on fitness. Again, a statistics module has been created to analyse length distributions.

3.2 Experiments

In [7] we provided strong theoretical and experimental evidence that standard sub-tree swapping crossover with uniform selection of crossover points pushes a population of a-ary GP trees, in the absence of selection, towards a limiting distribution of tree sizes of the form:

$$\Pr\{n\} = (1 - ap_a)\binom{an+1}{n}(1-p_a)^{(a-1)n+1}p_a^n \tag{1}$$

This is known as a Lagrange distribution of the second kind. $\Pr\{n\}$ is the probability of selecting a tree with n internal nodes and a is the arity of functions that can be used in the creation of an individual. The parameter p_a was shown to be related to μ_0, the mean program size at generation 0, and a according to the formula:

$$p_a = \frac{2\mu_0 + (a-1) - \sqrt{((1-a)-2\mu_0)^2 + 4(1-\mu_0^2)}}{2a(1+\mu_0)} \tag{2}$$

Equation (1) was generalised using the Gamma function: $\Gamma(n+1) = n!$ in [9] to enable mixed arity tree internal node distributions to be predicted:

$$\Pr_g\{n\} = (1 - \bar{a}p_{\bar{a}})\frac{\Gamma(\bar{a}n+2)}{\Gamma((\bar{a}-1)n+2)\Gamma(n+1)}(1-p_{\bar{a}})^{(\bar{a}-1)n+1}p_{\bar{a}}^n \tag{3}$$

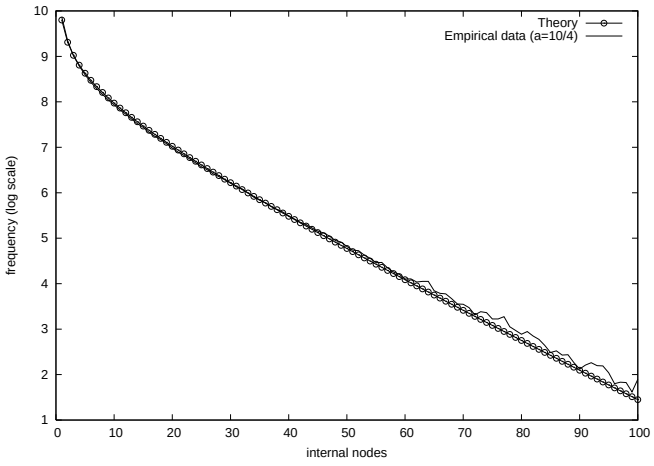

Fig. 2. Comparison between empirical and theoretical program size distributions for trees made up of a mix of 1, 2, 3 and 4 arity functions ($\bar{a}$=10/4) initialised with FULL method (depth = 3, initial mean size $\mu_0 = 25.38$, mean size after 500 generations $\mu_{500} = 23.76$)

$\bar{a}$ being an averaged arity of the primitive set. This can be calculated for mixed function arities from experimental initialisation parameters as follows:

$$\bar{a} = E(arity(F)) = \sum_{f} arity(f)P(F = f) \tag{4}$$

where E is the expectation operator, f is a non-terminal to be used in the GP experiment, $arity(f)$ is a function returning the arity of the non-terminal f,and $P(F = f)$ is the probability that a particular non-terminal f will be selected for a non-terminal node by the tree initialisation procedure. For traditional FULL and GROW initialisation methods non-terminals are chosen with equal probability.

For our first step towards extending Equation (3) to allow us to predict program length distributions with mixed arities, we look at what we can say for certain regarding the relationship between number of internal nodes and program length. First, we know for a-ary trees where the arities of all nodes in the tree are the same, the length, ℓ, of a program can be expressed exactly in terms of the number of its internal nodes, n, using the following equation:

$$\ell = a \times n + 1, \tag{5}$$

where a the (fixed) arity of the internal nodes. Therefore, rearranging Equation (5) to obtain internal nodes in terms of length, i.e.,

$$n = \frac{\ell - 1}{a}, \tag{6}$$

and substituting this into Equation (1), we obtain that, for a-ary trees,

$$\Pr_l\{\ell\} = \begin{cases} \Pr\{\frac{\ell-1}{a}\} & \text{if } \ell \text{ is a valid length (i.e., } \frac{\ell-1}{a} \text{ is a non-negative integer),} \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where $\Pr_l\{\ell\}$ is the limiting distribution of program lengths. This distribution applies, for example, for boolean function induction problems where often all functions are binary and symbolic regression problems where often only the standard four arithmetic operations are used.

Our next step is to extend the generalised formula for mixed-arity trees (Equation (3)) so as to predict length distributions rather than internal node distributions. We know that for a program length of 1, a single terminal, there will always be 0 internal nodes. Therefore, prediction of single node programs is a simple one-to-one mapping with the generalised formula for 0 internal nodes. However, other lengths can be obtained by different combination of internal nodes of different arities. For example, one can obtain programs of length 3 by using one internal node of arity 2 or two internal nodes of arity 1.

As a first approximation, we will assume that we can still estimate the expected number of internal nodes in a tree of length ℓ by applying Equation (6), simply using $\bar{a}$ instead of a. We can then substitute the result into Equation (3) to obtain the distribution of lengths we are looking for. Naturally, between the variable ℓ and the variable n there is a difference in scale (the factor $\bar{a}$). So, we will need to normalise the values produced by Equation (3) to ensure the new distribution sums to 1.

Putting all of this together, we obtain an approximate model of the limiting distribution of program lengths in the case of primitive sets of mixed arities. Namely:

$$\Pr_v\{\ell\} = \begin{cases} \Pr_g\{0\} & \text{if } \ell = 1, \\ \frac{\Pr_g\{\frac{\ell-1}{\bar{a}}\}}{\bar{a}} & \text{if } \ell \text{ is a valid length greater than 1.} \end{cases} \tag{8}$$

Note, we do not require $\frac{\ell-1}{\bar{a}}$ to be an integer.

By choosing to apply GP to a particular problem we have made an assumption that both fitness based selection and crossover are likely to provide an efficient search method to provide a solution. Normally a GP initialisation method is used GROW, FULL, RAMPED etc that takes no account of the eventual distributions 'desired' by either crossover or fitness (or any other GP operator such as mutation). Normally, eventual fitness values are not known in advance of a GP experiment, however, we now have evidence to show that crossover will, with repeated application, distribute the population according to a predictable distribution. Our next step is to investigate the possible benefits or disadvantages of initialising a population by length to take account of crossover.

We could of course write an algorithm to initialise the population according to the eventual predicted distribution desired by crossover. The most straightforward programmatic method, however, is to simply run the first 'X' number of generations of a GP experiment without fitness, thereby allowing crossover to distribute program lengths without having to endure the cost of fitness calculations.

The results of this experiment and verification of the above equations are described in the next section.

3.3 Results

In order to verify empirically the distributions proposed in 3.2 we performed a number of runs of a GP system in Java. A relatively large population of 100,000

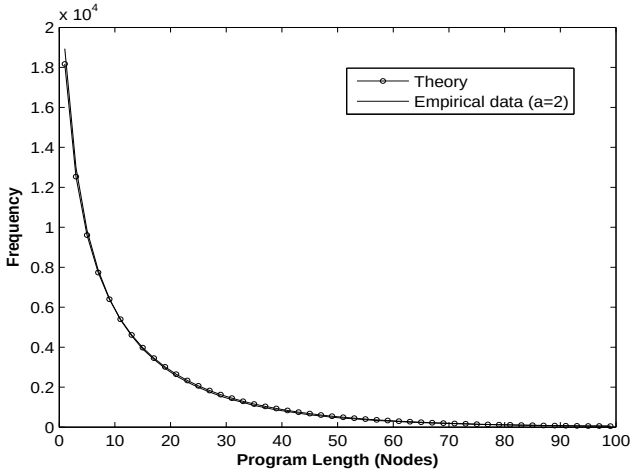

Fig. 3. Comparison between empirical and theoretical program length distributions for 2-ary trees initialised with FULL method (depth=3, initial mean size $\mu_0 = 15.0$, mean size after 500 generations $\mu_{500} = 14.49$). Invalid even lengths are ignored.

individuals was used in order to reduce drift of average program size and to ensure that enough programs of each length class were available. The FULL [8] initialisation method was used with each function having the same probability of selection. Each run consisted of 500 generations. A flat fitness landscape was used i.e. all fitness calculations returning a constant value.

Histograms were collected from the final generations (in order to ensure the effects of our chosen initialisation method have been washed-out[2]). For equation 3 each bin contained the number of internal nodes contained within a program. Figures 1 and 2 show our theoretical and observed results for arity mixes of 2, 2 & 3 and 1, 2, 3 & 4 respectively. Note, that in order to highlight the fit for larger and less common programs we used a log scale for frequency.

In order to verify our length distribution (equation 8) figure 3 shows an observed plotted length distribution for 2-ary trees, with invalid (even) lengths removed, compared to that predicted by $\Pr_l$. The observed values are again averages over twenty independent runs with populations of 100,000 individuals run for 500 generations. As we can see there is a very close fit between the two curves. Figure 4 shows the results of using arities of 1, 2, 3 & 4.

For our 'Lagrange-like' initialisation idea we took two out-of-the-box problems from the ECJ [11] evolutionary toolkit, the Artificial Ant (arities 2, 2 & 3) and 4 Bit Even Parity (arity 2 only), making adjustments to remove mutation, ensure uniform selection of crossover points, and to prevent a depth limit being applied. A population size of 1024 individuals was used and the results averaged over a hundred runs. All experiments were initialised using the FULL method with a

[2] It would of course be interesting to see how many generations are required to negate length effects of chosen initialisation methods. This has been left as a subject for future research.

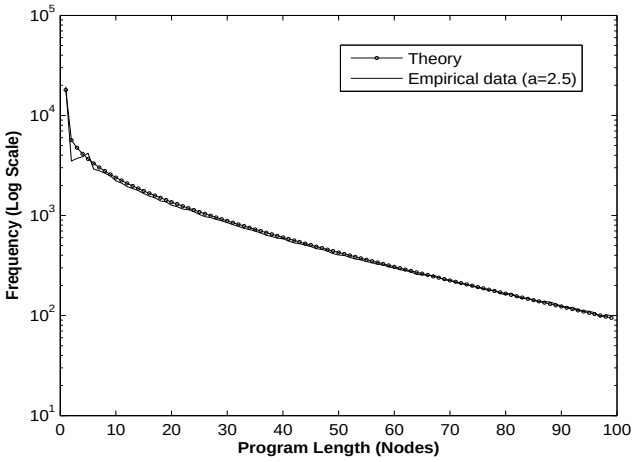

Fig. 4. Comparison between empirical and theoretical program size distributions for trees made up of 1, 2, 3 and 4 arity functions initialised with the FULL method (depth=3, μ_0 = 25.38, μ_{500} = 23.76). All lengths are valid.

depth of 3. Each experiment looked at the effect of running with zero, twenty or fifty initial generations where a constant fitness value was applied before allowing the experiment to continue as normal. Naturally, the flat-fitness phase was much faster than normal, since no fitness evaluation was required. During this phase crossover was free to distribute the population towards its limit size distribution.

Figures 5 and 6 show graphs of program growth for the Artificial Ant and 4-Bit Even Parity problems described. There is a noticeable increase in program growth for the populations with 'Lagrange-like' initialisation (via crossover-only prior generations).

At first one might find this result very surprising. How is it that initialisation has such a big effect on bloat, which is typically associated with late generations of a run, when effectively the population starts stagnating? An explanation for this lies in the combination of the crossover sampling distribution and fitness.

In every generation we produce a mating pool of relatively fit programs which will be used for reproduction. The standard crossover operator (with uniform selection of crossover points) will create a certain distribution of program sizes irrespective of the fitness of programs selected in that mating pool. For example, if size 45 programs are for some reason very fit, once in the mating pool, crossover takes no account of this fitness and produces a distribution of differing programs. From the theory provided earlier in the paper we know that this distribution is biased, in frequency, towards smaller programs, but with a tail of larger programs.

As the smaller programs will tend to be less fit (as a proportion of the population), these will not be selected for mating in the next generation. The larger programs will be selected as parents, thereby providing a higher probability of larger trees being created as children.

This explanation fits completely within the mathematical explanation for bloat provided in [12], where the following size evolution equation was derived[3]

$$E[\mu(t+1) - \mu(t)] = \sum_{l} N(G_l)(p(G_l, t) - \Phi(G_l, t)) \tag{9}$$

where E is the expectation operator, $\mu(t)$ is mean program size at time/generation t, G_l represents all programs of a particular shape, $N(G_l)$ represents the size of programs of shape G_l, $p(G_l, t)$ represents the selection probability for programs of shape G_l and $\Phi(G_l, t)$ represents their frequency in the population. As length classes are a super set of shape classes we can rewrite equation 9 as:

$$E[\mu(t+1) - \mu(t)] = \sum_{l} l \times (p(l, t) - \Phi(l, t)) \tag{10}$$

where $p(l, t)$ is the selection probability for programs of length l and $\phi(l, t)$ is their current proportion.

If, as is the case for the Ant and Parity problems, the selection probability for short programs is consistently lower than their frequency, then, everything else being equal, one must expect $E[\mu(t+1) - \mu(t)] > 0$, and hence bloat will occur.

In essence where a reasonable minimum size threshold exists for relatively fit programs, the repeated application of fitness based selection and standard crossover (with uniform selection of crossover points) to a population will cause bloat to occur. Our experimentation has shown that by allowing crossover to distribute a population before applying fitness we make this effect more acute. That is we are producing populations with proportionately shorter programs than those created using more traditional GP initialisation methods.

4 Achievements

We have presented within this paper a generalisation of the limiting distribution of program sizes for tree-based genetic programming when sub-tree swapping crossover is applied to a flat-fitness landscape. The formula provides evidence that the reproduction process has a strong bias to sample exponentially shorter programs. This bias is also most acute when a population has a small mean program size, refer to [9] for details [4].

A number of practical concerns have been presented in light of this bias. The first is an inability to sample large parts of the problem space i.e. those associated with larger programs. Additional problems with crossover are also encountered when fitness is associated with certain minimum program sizes leading to relatively poor performance in the finding of fit solutions and a tendency for the population to bloat.

[3] A major finding of this paper is that for symmetric subtree-swapping crossover operators e.g. standard crossover, 'The mean program size evolves as if selection only was acting on the population'. Derivation of this equation is explained in section 5.4 of [12].

[4] This paper also details the detrimental effect of fitness during a GP run when this bias is in full effect.

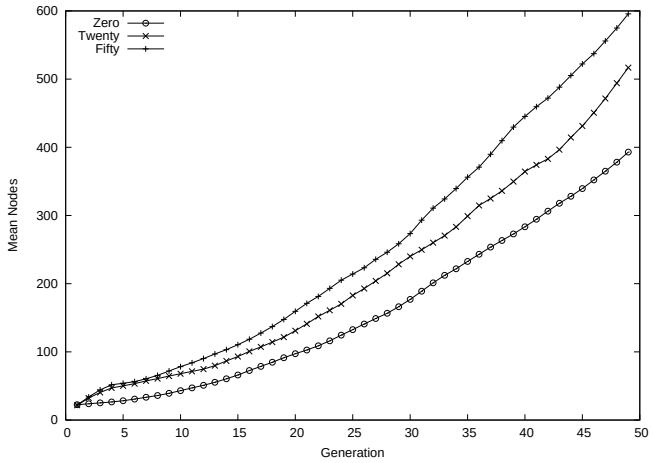

Fig. 5. Comparison of mean number of nodes per individual for populations with zero, twenty and fifty prior generations of crossover without fitness for the Artificial Ant problem

We have also shown that bloat can occur even from generation 0. It normally happens effectively only in much later generations because it takes time for crossover to skew the size distribution towards short programs, i.e. towards its limiting Lagrange distribution of the second kind.

It is therefore evident that future research should look into ways of modifying the bias of crossover if bloat is to be prevented. To do this we have created a new technique called Operator Equalisation (results to be published). This is a programmatically simple method that can be easily applied to current experimental environments that allows us to finely bias GP search to sample specific program lengths. In particular, the method can force GP to sample the search space using static (and arbitrary) length distributions. This completely and naturally suppresses bloat.

As we have shown the formulae presented show an exponentially increasing bias for crossover to sample smaller programs. As there are exponentially fewer unique smaller programs than larger programs, the sampling of new programs becomes less likely during a GP run if only crossover is applied. This bias becomes more acute with the application of a length limit (results to be published). The effect will become more prevalent as fitness values converge (i.e. as the fitness landscape flattens) during later stages of a GP run or when areas of neutrality are reached.

In order to explore what happens if one directly addresses this sampling-related cause for bloat, we have introduced a novel technique called Sampling Parsimony. By setting a penalty and a resampling maximum/minimum boundary this can be used to accelerate growth as well as to reduce its effect. We have not, however, directly verified if Selection Parsimony is competitive with other antibloat techniques and if so in which circumstances. We will, however, address this question in future work.

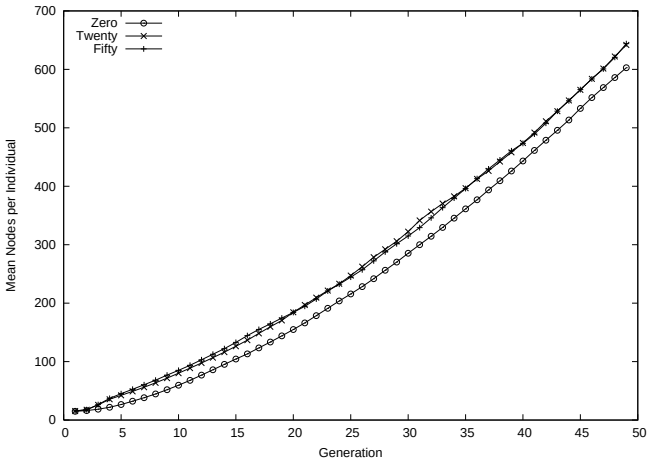

Fig. 6. Comparison of mean number of nodes per individual for populations with zero, twenty and fifty prior generations of crossover without fitness for the 4 Bit Even Parity problem

5 Feedback

I would be interested in any related work that may have or is currently being undertaken by researchers. I would also be interested in how this can be applied to other areas, for example my work on neutrality was a by-product of this research.

Of particular interest is the modeling of selection in conjunction with genetic operators. Current research notably in the use of schema theory has treated selection as a black box that cannot be opened. I have continued this trend in my own work. I would be interested, therefore, in any ideas regarding how to combine mathematical models with selection.

Finally, any advice with regard to experimental method would be much appreciated.

References

1. W. B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.
2. W. B. Langdon. How many good programs are there? How long are they? In Kenneth A. De Jong, Riccardo Poli, and Jonathan E. Rowe, editors, *Foundations of Genetic Algorithms VII*, pages 183–202, Torremolinos, Spain, 4-6 September 2002. Morgan Kaufmann. Published 2003.
3. W. B. Langdon. Convergence of program fitness landscapes. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1702–1714, Chicago, 12-16 July 2003. Springer-Verlag.
4. Riccardo Poli and Nicholas Freitag McPhee. Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application

to the study of the evolution of size. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 126–142, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.

5. Riccardo Poli and Nicholas Freitag McPhee. General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation*, 11(2):169–206, June 2003.
6. Sean Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, September 2000.
7. Riccardo Poli, William B. Langdon, and Stephen Dignum. On the limiting distribution of program sizes in tree-based genetic programming. In Marc Ebner, Michael O'Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 193–204, Valencia, Spain, 11 - 13 April 2007. Springer.
8. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
9. Stephen Dignum and Riccardo Poli. Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1588–1595, London, 7-11 July 2007. ACM Press.
10. A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.
11. Sean Luke. ECJ 16: A Java evolutionary computation library. http://cs.gmu.edu/∼eclab/projects/ecj/, 2005.
12. Riccardo Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 204–217, Essex, 14-16 April 2003. Springer-Verlag.

Evolving Tactical Teams for Shooter Games using Genetic Programming

Darren Doherty

Department Of Information Technology
National University Of Ireland, Galway.
darren.doherty@nuigalway.ie

Abstract. In recent years, there has been an emergence of squad-based shooter computer games. For a team to be tactically proficient, intelligent non-playable characters (NPCs) must be created that are able to assess their situation, choose effective courses of action and coordinate their behaviour so that they work together effectively. This is a very difficult task and game developers are still striving to create teams of NPCs that are able to display effective team behaviours. Our research examines genetic programming (GP) as a technique to automatically develop effective team behaviours for shooter games. Previous experiments have given rise to GP evolved teams capable of consistently defeating a single powerful enemy agent. The behaviours of these teams have been analysed using a technique we developed for analysing team phenotypes. In future work, we wish to incorporate explicit communication into the evolution and improve our phenotypic analysis method.

1 Introduction

1.1 Problem and Open Questions

Over the past five years, there has been an emergence of more squad-based shooter computer games. The non-playable characters[1] (NPCs) in these games are required to be tactically proficient by working together intelligently to overcome their enemy. Rather than attempting to develop complex behavioural systems that allow the NPCs to display intelligent team behaviour, game developers are still using deterministic techniques to implement the artificial intelligence (AI) of the individual NPCs and have opted to use simple techniques to make it appear as if the NPCs are cooperating in an intelligent manner. For example, some game developers prevent two NPCs from shooting at the player simultaneously, causing them to appear to be taking turns attacking the player. This is combined with audio cues from the NPCs such as shouting "cover me" when an NPC goes to reload its weapon to create the illusion of cooperative behaviour. However, using rudimentary or "cheating" mechanisms to simulate cooperative behaviour in squad-based shooter games is less than ideal. Moreover, the use of deterministic techniques results in repetitive and predictable behaviour.

[1] A non-playable character is any agent in the game not controlled by a human player.

In order for NPCs to work together effectively they must display tactical team behaviour. Team tactics can be defined as the set of methods used by a team for attaining their common goals. As tactics are highly dependent on the situation [1] (i.e. terrain, team supplies, enemy movement, etc) it is very difficult for game developers not only to hardcode the team tactics themselves into the AI of the NPCs (commonly referred to as game-AI) but also to decide when and where it would be effective to use certain tactics. Developers must create an AI architecture that allows a team of NPCs to communicate and coordinate their behaviour in an intelligent manner, such that they display effective group rational behaviour. This task is by no means trivial as achieving coordination among multiple autonomous agents in any domain can be a difficult task and shooter games are set in very complex environments. As such, developers of shooter games are still striving to create teams of NPCs that are able to correctly assess a situation, choose effective courses of action for each team member and work together to achieve their common goal by displaying effective team behaviours. We propose that evolutionary computation (EC) techniques, in particular genetic programming (GP), have the ability to discover effective and novel team behaviours automatically and that these behaviours can be formally analysed to show that they are tactically proficient.

There are a number of open research questions that we aim to answer:

1. Can genetic programming be used as a technique to successfully evolve effective and novel tactical behaviours for teams of non-playable characters in shooter games?
2. Can the behaviours of evolved teams be formally analysed to discover the team behavioural characteristics required for teams to perform effectively?
3. Does explicit communication between the team agents encourage more cooperative team behaviours and allow for more effective team behaviours to emerge from the evolutionary process?

1.2 Related Work

In the past, EC techniques have been comprehensively applied to board games (e.g. checkers [2]) and card games (e.g. blackjack [3, 4]) in attempts to find optimal playing strategies. However, EC techniques have not been used as extensively in the exploration and research of AI in computer games. In general, the environment and agent behaviours in a computer game are very complex. Thus, developers have been hesitant to introduce EC techniques into their AI systems as there is a fear the evolutionary algorithm will not find good solutions. Nevertheless, a small number of games have taken the risk of incorporating EC into their game-AI and have proven to be very successful. Such games include: *Black & White* [5] and *S.T.A.L.K.E.R.: Shadow of Cherynobyl* [6]. As EC techniques are very resource intensive and can take a long time to produce solutions, they are more suited to offline game-AI development (i.e. during game development) as opposed to online (i.e. evolving while the game is being played). The research community has begun to realise the potential of EC techniques as developmental

tools for game-AI. Champandard [7] used a GA to successfully evolve NPCs in an first-person shooter game to dodge enemy fire. Ponsen et al. [8] have used GAs off-line to design tactics for an RTS game. Cole et al. [9] used a GA to successfully tune an NPC's weapon selection parameters in a shooter game.

A few attempts have been made at evolving teams of NPCs for shooter games [10, 11]. However, neither of these methods are ideal for game developers to use to create tactically sound behaviours. In both cases, the team's behaviour is evolved in an adaptive manner, while the game is being played (i.e. online), so developers cannot tell or test, in advance, what behaviours the NPCs will exhibit or how tactically proficient they will be. Moreover, in the first system [10], a human player is required to specify, which NPC attributes are to be evolved and the second mechanism [11] requires a number of game-specific enhancements to the GA paradigm in order to work, such as a history fallback concept.

GP has been successfully used to simulate team evolution in a number of different simulated domains. GP was first applied to team evolution by Haynes et al. [12]. Luke and Spector [13] used GP to successfully evolve predator strategies that enable a group of lions to successfully hunt gazelle. In Luke's work, heterogeneous teams are shown to perform better than homogeneous teams. GP has also been used to enable a team of ants to work together to solve a food collection problem [14]. The ants must not only cooperate in order to reach the food but must also work together to carry it as it is too heavy for one ant to carry alone. Richards et al. [15] used a genetic program to evolve groups of unmanned air vehicles to effectively search an uncertain and/or hostile environment. The environments used in Richard's experiments were relatively complex, consisting of: hostile enemies, irregular shaped search areas and no fly zones. In addition, GP has been used to successfully evolve sporting strategies for teams of volleyball players [16] and teams of soccer players [17]. We hypothesise that GP can be used to evolve effective and novel team behaviours for shooter games in a similar manner as has been done in other domains.

In the past, there has been little research on how to compare evolutionary computation solutions at a phenotypic level. Typically, phenotypes are compared at a fitness level [18]. However, this is a very crude measure and is not useful in more complex environments. Another, more expressive approach is to compare phenotypes at a behavioural level by recording agents responses to stimuli and comparing the responses using a distance metric [19]. As the team chromosomes evolved in our work are quite complex, a simple distance metric cannot be used to analyse their behaviour. Therefore, we must develop our own phenotypic analysis method that allows for the formal analysis of team behaviours. This technique need not be constrained to our research alone and could potentially be used to analyse team behaviours in any domain that involves cooperative teamwork.

In a study conducted by Barlow et al. [20], it was found that role assignment, communication and coordination are the three main factors that contribute to a team's success. Moreover, Best and Lebiere [21] state that communication is a necessary prerequisite to teamwork. In the past, GP has been used to successfully evolve team behaviours when there was no explicit communication between team

members [13, 16, 22]; the agents coordinate their behaviour and cooperate by evolving to do so implicitly. To this end we wish to investigate the effect, if any, explicit communication between the team members has on teamwork and if it encourages the evolution of more effective team behaviours.

2 Research and Study

2.1 Goals

The following hypotheses are presented in this research:

Hypothesis 1

> "Genetic Programming can be used to develop effective and novel tactical behaviours for teams of non-playable characters in shooter games."

An "effective" tactical behaviour can be described as any behaviour that enables the team to achieve their common goal on a consistent basis. In the case of shooter games, the main goal of the team is to defeat their common enemy so the team's effectiveness can be measured as its win percentage[2] of games played. A "novel" tactical behaviour can be described as any behaviour that is new or different to those behaviours that are already known to exist.

Hypothesis 2

> "An analysis of the evolved team behaviours will reveal the behavioural characteristics required for teams to perform effectively."

A phenotypic analysis will be performed on evolved teams in order to assess their behaviours. The ability to analyse the behaviours of teams should aid game developers in the understanding and design of effective team behaviours by uncovering what attributes are needed for teams to perform effectively. In order to fully test this hypothesis, the behavioural characteristics that are extracted from this process will be used to manually create a new team which will then be tested for effectiveness.

Hypothesis 3

> "Explicit communication between the team agents should allow for more effective team behaviours to emerge from the evolutionary process."

Explicit communication will be used to enable agents to share perceived information (such as the position of the enemy or items) and to send and receive tactical commands (such as telling a teammate to provide covering fire or to retreat from a position) to assess the effect, if any, explicit communication between team agents will have on the effectiveness of the teams that are evolved.

[2] Draws are not counted in the measure of effectiveness as they are very uncommon.

2.2 Current Status

Some of the above goals have been partially achieved in our on-going research[23–25]. In previous work, we successfully evolved a team of NPCs to defeat a single powerful enemy agent using GP [23]. This type of environment is akin to single player shooter games where the single enemy agent can be viewed as the human player playing against a team of NPCs (i.e. the evolved team). The enemy agent uses hand-coded desirability algorithms to rationally choose which goal to pursue (see section 3), has a health level equal to the entire team of agents and begins the game with the strongest weapon. This work was extended to evolve teams against the enemy agent in environments of varying difficulty [25]. In these experiments, we show that the difficulty of the environment has a bearing on the ability of the genetic program to evolve effective teams. Our first goal is partly accomplished as we have shown that it is possible to evolve effective team behaviours for shooter games using GP [23, 25]. These behaviours have been assessed over 1000 games to get a measure of their effectiveness (see section 3.2).

Part of the second goal has also been accomplished as we developed a technique that allows for the analysis of evolved team behaviours by comparing teams at a phenotypic level [24]. Our technique successfully extracts common behavioural traits of teams that perform similarly. Thus, it can be used to reveal the behavioural characteristics that are required for teams to perform effectively. However, the technique has limitations and can be improved. The third goal has not yet been addressed.

2.3 Future Plans

Our first goal is partly accomplished as we have shown that it is possible to evolve effective team behaviours for shooter games using GP. However, we have not yet tested the generalisability or novelty of these evolved behaviours. Thus, testing these will be our immediate tasks for future work.

Our second goal has also been partly accomplished as we have created a technique that enables us to successfully analyse evolved team behaviours. However, it does have limitations. We plan to improve the phenotypic analysis technique used in [24] in order to more accurately analyse the team behaviours.

Our third goal will involve two communication experiments. The first will examine the effect, if any, that sharing perceptual information between team agents has on the effectiveness of evolved teams. In our second experiment, explicit communication nodes will be added to the genetic program in an attempt to evolve effective communication of both perceptual information and tactical commands. Here, communication can only emerge through the evolutionary process as agents must evolve to send messages and respond appropriately to messages.

When my PhD is completed, I would like to stay in academia and continue research of EC and game-AI.

2.4 Study

My PhD duration is four years and I am currently in my third year of study.

3 Results

The environment is a 2-dimensional space, enclosed by four walls and is built using the *Raven* game engine [26]. Items are placed on the map at locations equidistant from both the team starting points and the enemy starting point. These items consist of health packs and a range of weapons that respawn after a set time if collected (see Fig. 1).

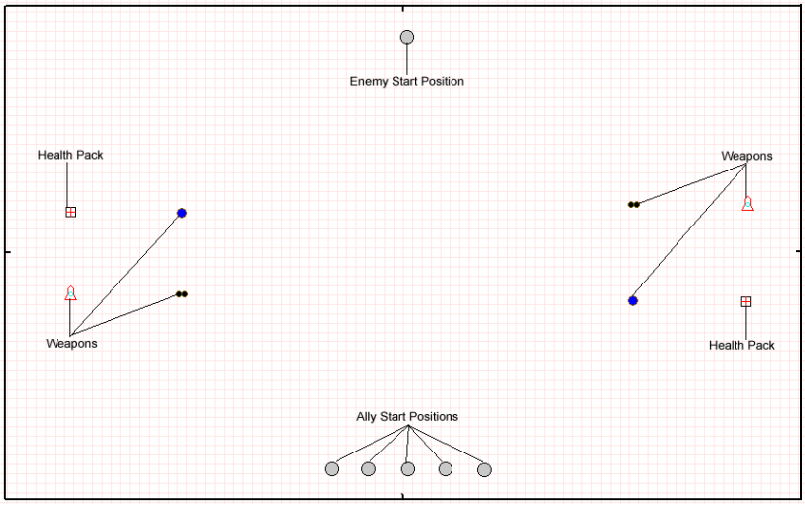

Fig. 1. Simulation environment map

Both types of agent (i.e. team agents and enemy agent) use the same underlying goal-driven architecture [27] to define their behaviour. Composite goals are broken down into subgoals, hence a hierarchical structure of goals can be created. Goals are satisfied consecutively so the current goal (and any subgoals of it) are satisfied before the next goal is evaluated. If an NPC's situation changes, a new, more desirable goal can be placed at the front of the goal-queue. Once this goal is satisfied, the agent can continue pursuing its original goal. Although the underlying goal architecture is the same, team agents use a decision-making tree evolved using GP to decide which goal to pursue, whereas the enemy agent uses desirability algorithms associated with each goal. The desirability algorithms are hand-coded to give the enemy intelligent reasoning abilities.

–

$$Desirability_{GetHealth} = a \times \left(\frac{1 - Health}{Distance\_to\_Health} \right)$$

where a is a random bias, $Distance\_to\_Health$ is the distance to the nearest health pack (if known) and $Health$ is the agent's current health. If $Distance\_to\_Health$ is unknown the desirability returned will be zero.

–
$$Desirability_{Attack} = b \times Total\_Weapon\_Strength \times Health$$
where b is a random bias and $Total\_Weapon\_Strength$ is the proportion of ammunition the agent has for all weapons. The agent will only attack if its attacking ability (i.e. $Total\_Weapon\_Strength$) and health are both high.
–
$$Desirability_{GetWeapon} = c \times \left(\frac{Health \times (1 - Weapon\_Strength)}{Distance\_to\_Weapon} \right)$$
where c is a random bias, $Weapon\_Strength$ is the proportion of ammunition the agent has for the weapon and $Distance\_to\_Weapon$ is the distance to the nearest ammunition pack for that weapon (if known). If $Distance\_to\_Weapon$ is unknown the desirability returned is zero. Each weapon for which ammunition can be retrieved has an associated desirability algorithm.
–
$$Desirability_{Explore} = d$$
where d is a constant. An agent will only explore if all other desires are low.

The random biases cause the enemy's behaviour to vary from game to game.

Both types of agent have a memory allowing them to remember information they perceive. Any dynamic information, such as ally or enemy positions, is forgotten after a specified time. If more than one team agent has been recently sensed by the enemy, the enemy will select its target based on distance. Weapons have different ideal ranges within which they are more effective and bullets for weapons have different properties, such as velocity, spread, etc.

3.1 Goal 1 - Evolving Teams

The entire team is viewed as one chromosome, so fitness, crossover and mutation operators are applied to the team as a whole. A chromosome comprises five GP subtrees (one for each agent), so evolved teams are heterogenous (see Fig. 2).

Evolutionary runs have been completed in environments where agents had perfect vision through a 180 degree field of view [23] and in environments where team agents have restricted viewing distances of 50, 200, 350 and 500 pixels and fields of view of 90 and 180 degrees [25]. In the latter work, the enemy has twice the viewing distance of a team agent but their fields of view are identical. Twenty runs are completed in each of the environments and 100 chromosomes are evolved over 100 generations in each of the runs.

Node Sets A strongly typed genetic program [28] is used in order to constrain the type of nodes that can be children of other nodes. There are five node sets:

- Action nodes represent goals the agent can pursue and the *IF* statement.
- Condition nodes represent conditions under which goals are to be pursued.
- Position nodes represent positions on the map to which the agents can move.
- Environment nodes represent gaming parameters that are checked during the agent's decision making process (e.g. ammunition supplies).
- Number nodes represent arithmetic operators and constants.

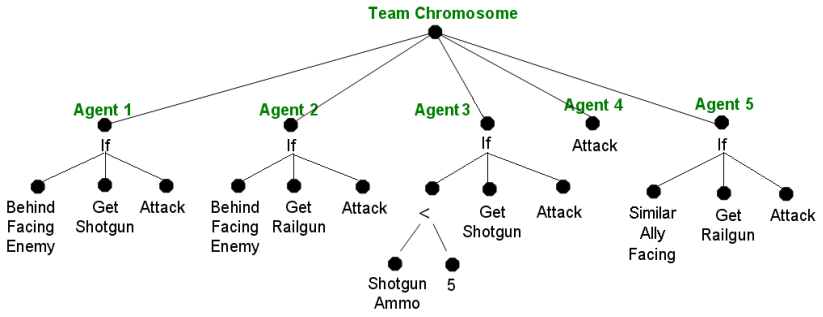

Fig. 2. Sample GP chromosome

Fitness Calculation The fitness function takes into account: the remaining health of both the enemy and team agents, the duration of the games and the length of the chromosome (to prevent bloat).

$$RawFitness = \left(\frac{(EW \times (Games \times TSize \times MaxHealth - EH) + AH)}{Games \times TSize \times MaxHealth}\right) + \left(\frac{AvgGameTime}{Scaling \times MaxGameTime}\right)$$

$$StdFitness = (MaxRawFitness - RawFitness) + \left(\frac{Length}{LengthFactor}\right)$$

where $Games$ is the number of games played per evaluation (i.e. twenty), $TSize$ is the team size (i.e. five), $MaxHealth$ is the maximum health a team agent can have (i.e. fifty), EH and AH are the total amount of health remaining for the enemy agent and for all five team agents respectively, EW is a weight (set to five) that gives more importance to EH than AH, $AvgGameTime$ is the average duration of the games, $Scaling$ reduces the impact game time has on fitness (set to four), $MaxGameTime$ is the maximum duration of a game (i.e. 5000 game updates), $MaxRawFitness$ is the maximum value $RawFitness$ can hold, $Length$ is the length of the chromosome, $LengthFactor$ is used to limit the influence $Length$ has on fitness (set to 5000) and $StdFitness$ is fitness standardised such that values closer to zero are better.

Selection, Crossover and Mutation There are two forms of selection used. The first is a form of elitism where m of the best n chromosomes from each generation are reproduced unchanged into the next generation. The second method is roulette wheel selection. Any chromosomes selected in this manner are subjected to crossover and mutation (given crossover probability of 0.8 and mutation probability of 0.1 respectively). To increase diversity, there is also a 2% chance for new chromosomes to be created and added to the population each generation.

The crossover operator is specifically designed for team evolution [29]. A random $Tsize$ bit mask is selected that decides which of the team agents in the parent chromosomes are to be altered during crossover. A '1' in the mask indicates that the agent at that position is copied directly into the child and a '0' indicates that the agent is to take part in crossover with the corresponding agent of the other parent. A random crossover point is then chosen within each agent to be crossed over. The node at the crossover points of corresponding agents must be from the same node set in order for the crossover to be valid.

Two forms of mutation are used. The first, swaps two randomly selected subtrees between agents in the same chromosome. Similar to crossover, the root nodes of the subtrees must be from the same set. The second form randomly selects a subtree from the chromosome and replaces it with a newly created tree.

Testing for Effectiveness As mentioned in section 2.1, the effectiveness of a team behaviour can be measured by the team's win percentage. The behaviour of the enemy will be slightly different in each game due to the random biases used when initialising its desirability algorithms. In order to get an accurate measure of a team's effectiveness, 1000 games are played per team evaluation and the percentage of wins recorded. To demonstrate the effectiveness of evolved teams, their performance is compared to the performance of a team of generic agents with the same AI as the enemy.

To show that the genetic program gives a strong result, the twenty evolved teams from each environment are compared against the twenty best teams obtained from a random search of that environment. The sample size used for the random search is 10000. The random teams are evaluated over 1000 games. The performance of the evolved teams is compared to the performance of the random teams using paired T-tests with a confidence interval of 95% to ascertain whether the differences are statistically significant.

Generalisability of Effective Teams Although, the test for team effectiveness shows that teams are somewhat generalisable against variations of the enemy in their native environment it does not show if they are generalisable to other enemies and environments. To examine the generalisability of behaviours, the effectiveness of each team must be assessed in a range of environments and against different types of enemy. The environment can be changed by altering the size and shape of the map and the location and quantities of items. The enemy can be changed by altering the way in which it selects its target or by giving it greater capabilities such as increased health or speed. A team behaviour can be said to be generalisable if it can be shown to have a similar effectiveness in a range of environments and against a range of enemies.

Testing for Novelty In order to test for novelty, we must first have an existing behaviour set in place to which we can compare the evolved behaviours. There will be 100 behaviours in the behaviour set that comprises generic and

random team behaviours. Generic behaviours are derived from the same AI that is used by the enemy agent in the game. All agents on the team use desirability algorithms to choose their goals. Different biases are used on the desirability algorithms to obtain a variety of generic team behaviours. Random behaviours are the best behaviours obtained from a random sample of the set of all behaviours.

One way to measure the novelty of a team behaviour is to check if its performance is statistically different to the performance of all the known behaviours. To obtain a robust measure of performance, 1000 games are simulated for each behaviour and a paired T-test is performed between the evolved behaviours and the known set of behaviours using a confidence interval of 95%. If the performance of the evolved behaviours are shown to be significantly different to all the known behaviours then the evolved behaviours are novel as they are different to the known set. An alternative measure of novelty is presented in section 3.3.

3.2 Results - Goal 1

In the experiments where agents have perfect vision through a field of view of 180 degrees, a number of teams emerged from the evolutionary runs capable of consistently defeating the enemy agent [23]. When team effectiveness is tested, the best evolved team attained a win percentage of 94% over the 1000 games. An additional experiment was carried out to compare the performance of best evolved team against the performance of a team of generic agents with the same AI as the enemy. Each team was evaluated over 1000 games to obtain a robust measure of their effectiveness. In this experiment, the evolved team won 943 of the 1000 games, whereas the generic team only won 411 of the 1000. The evolved team significantly outperforms the team of five individually rational agents.

In another set of experiments, evolutionary runs were completed in environments of varying difficulty [25]. The results show that the level of difficulty of the environment does have an impact on the genetic program's ability to evolve effective team behaviours. The results follow a similar trend for both, the environments where the field of view is 90 degrees and environments where the field of view is 180 degrees. Paired T-tests performed between the resulting groups of teams from each environment show a statistically significant increase in fitness of evolved teams as the viewing distance of agents increase for a confidence interval of 95%. The number of games won (out of 1000) by the best evolved team in each of the environments were also recorded. Generally, as the level of difficulty of the environment decreases, the number of games won by the best evolved team increases. In the most difficult environments, where the viewing distance of agents is 50 pixels, the maximum win percentage for any of the teams is less than 8% of games played. In contrast, the win percentage of the best evolved team in the least difficult environment, is over 97% of games played.

To test the performance of the genetic program, the twenty evolved teams from each environment are compared to the twenty best teams obtained from random search in each environment. The results show that the evolved teams are statistically better than those obtained from random search with a confidence interval of 99% in all environments.

3.3 Goal 2 - Phenotypic Analysis

Our phenotypic analysis approach classifies the behaviours of evolved teams by grouping teams based on their performance and ascertaining the behavioural traits common to like-performing teams. Agent types are identified that encapsulate the different behaviours agents can display in the game. These types include: decoys, rocket ammunition gatherers, shotgun ammunition gatherers, railgun ammunition gatherers, rocket attackers, shotgun attackers, railgun attackers, blaster attackers, evaders, health preservers and cohesive agents. Each type has specific behavioural traits but types are not mutually exclusive.

1000 games are simulated for each team in which periodic snapshots of the agents' game state are taken in order to capture their behavioural information. The agent types have associated degree of membership algorithms that are used to calculate the degree of membership each agent has to every agent type by analysing the captured behavioural information of that agent. Each team agent receives a probability score for each agent type that indicates how likely their behaviour describes each type. When these probability scores have been calculated for every team agent they are combined into a single team vector describing the overall team's behaviour. In order to combine agent probability scores into the team vector, four different probability ranges are defined for each agent type such that the number of fair, good, very good and excellent agents of each type on the team can recorded.

The team vectors are then put into a decision tree classification algorithm to be analysed to see if there is a correlation between a team's performance (fitness score) and the combined behaviours of the team's agents. We choose to select 10-fold cross validation for our decision tree analysis to give more accurate results as our datasets are relatively small (size twenty). The path from the root node of the decision tree to the leaves should reveal the behavioural characteristics that are common to teams who perform similarly and the combination of agent types on a team that allow for good team performance. The performance of each group of teams in the leaf nodes of the tree are tested to determine if there is a statistical difference in their performance for a confidence interval of 95%. In doing this, it can be said that the different behavioural characteristics of each group of teams lead to a significantly different team performance.

Testing for Novelty In this section, we present an alternative method of determining behaviour novelty to the one outlined in section 3.1. A team behaviour can be regarded as novel if an analysis of that behaviour shows that the team displays different behavioural traits to those of the known set of team behaviours. A technique has been specified that enables the discovery of behavioural characteristics of teams. To determine the similarity of two team behaviours, a technique must now be specified to compare the teams based on their behavioural traits.

A technique has been specified that creates a single vector describing the team's behaviour. Each point in the vector represents the degree to which agents on the team fulfill the role of a particular agent type. By calculating the difference between the corresponding points on two team vectors and summing up

the absolute values of the resulting differences, a value can be obtained that ascertains how similar or different the two teams are based on their behaviour. The higher the value for all comparisons between the evolved behaviour and the set of known behaviours, the higher the degree of novelty.

3.4 Results - Goal 2

The results of our phenotypic analysis of teams evolved in [23] show that teams with comparable fitnesses have common behavioural traits [24]. The most effective teams were shown to have two or more strong attacking agents and one or two decoy agents. The decoys distract the enemy, whilst the other offensive team members collect ammunition and weapons before attacking the enemy simultaneously. The actions of the decoy agents in these teams do not appear individually rational but their behaviour is essential to the success of the team. To test the validity of our approach, a team has been manually created that encompasses the behavioural traits of the most effective teams and its performance assessed. This team scored a win percentage of 87% over 1000 games.

The teams recorded from our experiments in environments of varying difficulty [25] have also been formally analysed but the results have yet to be published. The results show that the less effective teams emerging from the most difficult environments have very little attacking ability with the best teams from theses environments containing only agents that attack with the weakest weapon. In contrast, the better performing teams that emerged from the least difficult environments show clear signs of emergent teamwork. These teams have at least two very strong attackers as well as two or more good decoy agents to draw away enemy fire from the attackers. These results compliment those in [24].

4 Achievements and Future Work

Our results show that GP can be used to develop effective team behaviours in shooter games. The evolved behaviours have been shown to be robust and somewhat generalisable. A phenotypic analysis of these behaviours shows that effective teams share common traits. In addition, these evolved behaviours have been shown to outperform a team of generic agents in the same environment.

Our immediate tasks for the future will be to explore the generalisability and novelty of evolved teams as set out in sections 3.1, 3.1 and 3.3 respectively. We would also like to improve on our phenotypic analysis technique as it has limitations. When the communication experiments are complete the phenotypic analyses of teams may reveal how important a role communication plays in a team's performance. The bulk of our future research will involve exploring the effects communication between agents has on the evolution of team behaviours. Agents could evolve to share perceived information and to send messages and respond accordingly to messages received, which could allow for much more effective team behaviours to evolve.

5 Feedback

The Doctoral Consortium will provide me with a chance to get feedback on my research from leading experts in the field of GP. I would like to know how relevant or interesting they find my research, in its contribution to the scientific and gaming communities. I am confident they will provide me with constructive criticism that will help me along the way to achieving my research goals.

Acknowledgement

The author would like to acknowledge the Irish Research Council for Science, Engineering and Technology for their assistance through the Embark initiative.

References

1. Thurau, C., Bauckhage, C., Sagerer, G.: Imitation learning at all levels of game–AI. In: Proceedings of the 5th International Conference on Computer Games, Artificial Intelligence, Design and Education. (2004) 402–408
2. Chellapilla, K., Fogel, D.B.: Evolving an expert checkers playing program without using human expertise. IEEE Transactions on Evolutionary Computation **5**(4) (2001) 422–428
3. Kendall, G., Smith, C.: The evolution of blackjack strategies. In: Proceedings of the 2003 Congress on Evolutionary Computation (CEC2003). Volume 4., IEEE Press (2003) 2474–2481
4. Curran, D., O'Riordan, C.: Evolving Blackjack Strategies Using Cultural Learning. In: Adaptive and Natural Computing Algorithms. Springer Vienna (2005) 210–213
5. Lionhead-Studios: Black & White. (2001) http://www.lionhead.com/bw/.
6. GSC-GameWorld: S.T.A.L.K.E.R.: Shadow of Chernobyl. (2006) http://www.stalker-game.com/en/.
7. Champandard, A.J.: AI Game Development: Synthetic Creatures with Learning and Reactive Behaviours. New Riders Publishing (2004)
8. Ponsen, M.: Improving adaptive game–AI with evolutionary learning. Master's thesis, Delft University of Technology (2004)
9. Cole, N., Louis, S., Miles, C.: Using a genetic algorithm to tune first–person shooter bots. In: Congress on Evolutionary Computation 2004. Volume 1. (2004) 139–145
10. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Evolving neural network agents in the nero video game. In: Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG05). (2005)
11. Bakkes, S., Spronck, P., Postma, E.O.: Team: The team-oriented evolutionary adaptability mechanism. In Rauterberg, M., ed.: Proceedings of the Third International Conference on Entertainment Computing (ICEC 2004). Volume 3166 of Lecture Notes in Computer Science., Springer (2004) 273–282
12. Haynes, T., Wainwright, R., Sen, S., Schoenefeld, D.: Strongly typed genetic programming in evolving cooperation strategies. In Eshelman, L., ed.: Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95), Pittsburgh, PA, USA, Morgan Kaufmann (1995) 271–278

13. Luke, S., Spector, L.: Evolving teamwork and coordination with genetic programming. In Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L., eds.: Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, MIT Press (1996) 150–156
14. LaLena, M.: Teamwork in genetic programming. Master's thesis, Rochester Institute of Technology, School of Computer Science and Technology (1997)
15. Richards, M.D., Whitley, D., Beveridge, J.R., Mytkowicz, T., Nguyen, D., Rome, D.: Evolving cooperative strategies for UAV teams. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, New York, NY, USA, ACM Press (2005) 1721–1728
16. Raik, S., Durnota, B.: The evolution of sporting strategies. In Stonier, R.J., Yu, X.H., eds.: Complex Systems: Mechanisms of Adaption, Amsterdam, Netherlands, IOS Press (1994) 85–92
17. Luke, S., Hohn, C., Farris, J., Jackson, G., Hendler, J.: Co–evolving soccer softbot team coordination with genetic programming. In: International Joint Conference on Artificial Intelligence–97 First International Workshop on RoboCup, Nagoya, Japan (1997)
18. Burke, E.K., Gustafson, S., Kendall, G.: Diversity in genetic programming: An analysis of measures and correlation with fitness. IEEE Transactions on Evolutionary Computation **8**(1) (2004) 47–62
19. Curran, D., O'Riordan, C.: Increasing population diversity through cultural learning. Adaptive Behavior – Animals, Animats, Software Agents, Robots, Adaptive Systems **14**(4) (2006) 315–338
20. Barlow, M., Luck, M., Lewis, E., Ford, M., Cox, R.: Factors in team performance in a virtual squad environment. In: SimTecT 2004 Simulation Technology and Training Conference. (2004)
21. Best, B.J., Lebiere, C.: Spatial plans, communication and teamwork in synthetic MOUT agents. In: Proceedings of the 12th Conference on Behavior Representation in Modelling and Simulation. (2003)
22. Haynes, T., Sen, S.: The evolution of multiagent coordination strategies. Adaptive Behavior (1997)
23. Doherty, D., O'Riordan, C.: Evolving tactical behaviours for teams of agents in single player action games. In: Proceedings of the 9th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games. (2006) 121–126
24. Doherty, D., O'Riordan, C.: A phenotypic analysis of GP–evolved team behaviours. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, New York, NY, USA, ACM Press (2007) 1951–1958
25. Doherty, D., O'Riordan, C.: Evolving team behaviours in environments of varying difficulty. In: Proceedings of the 18th Irish Artificial Intelligence and Cognitive Science Conference. (2007)
26. Buckland, M.: Raven: An Overview. In: Programming Game AI by Example. Wordware Publishing, Inc (2005) 295–333
27. Orkin, J.: Applying Goal-Oriented Action Planning to Games. In: AI Game Programming Wisdom 2. Charles River Media (2004)
28. Montana, D.J.: Strongly typed genetic programming. Evolutionary Computation **3**(2) (1995) 199–230
29. Haynes, T., Sen, S., Schoenefeld, D., Wainwright, R.: Evolving a team. In Siegel, E.V., Koza, J.R., eds.: Working Notes for the AAAI Symposium on Genetic Programming, Cambridge, MA, AAAI (1995)

Optimization of the Operation of Pipeless Plants by an Evolutionary Algorithm and Simulation

Sabine Piana

Graduate School of Production Engineering and Logistics
Process Dynamics and Operations Group
Technische Universität Dortmund, Germany
Sabine.Piana@bci.tu-dortmund.de

Abstract. During the operation of pipeless plants, decisions have to be made on the scheduling of the production, on the assignment of the equipment and on the routing of the vessels. Many interacting degrees of freedom prohibit the use of exact mathematical algorithms to compute optimal schedules. We propose an evolutionary scheduling algorithm that uses a simulator with internal heuristics as a schedule builder in the evaluation of the fitness of the individuals. Repair algorithms eliminate most infeasibilities before passing a candidate solution to the schedule builder. The algorithm yields up to 17 % shorter makespans than the as-soon-as-possible (ASAP) solution.

1 Introduction of the Research Area

Many real-world planning and scheduling problems are too complex to be treated by exact approaches. It is nonetheless often possible to model the problem at a high level of detail and to simulate it by appropriate tools. In practice, "optimization" is then often done by extensive simulation studies. Due to the large number of degrees of freedom, it is, however, unlikely that a good solution is found by this approach. A promising approach is to combine simulation with optimization in a way where some degrees of freedom are varied by the optimization algorithm whereas others are handled directly by the simulation which often includes heuristics to solve subproblems. Evolutionary algorithms (EAs) are particularly suited for this kind of hybrid optimization as they can embed simulations as black-box computations of cost function values.

1.1 Optimization of Pipeless Plants

Pipeless plants are an alternative to traditional multiproduct batch plants with fixed piping. Their distinctive feature is that the chemical processing steps are performed at stationary processing stations and that the substances are moved around in mobile vessels by automated guided vehicles (AGVs) instead of pumping them through pipes. The sequences of the production steps are defined by the recipes that determine the order in which a vessel must visit the different

processing stations. Pipeless plants offer a high degree of flexibility. A change of the priorities of the orders can be realized by re-directing vessels. The reduction of fixed piping results in significantly less time and effort for cleaning and sterilizing when a product changeover occurs. The cleaning of the vessels is carried out in separate cleaning stations and the processing stations are cleaned in place. It is not necessary to shut down a number of coupled units of the plant during cleaning cycles.

The increased flexibility of a pipeless plant comes at the expense of a higher degree of complexity. Computer-aided operation is indispensable for pipeless plants. During plant operation a set of coupled problems has to be solved. Decisions have to be made on the scheduling of the production orders, on the assignment of orders to vessels, stations and AGVs, and on the conflict-free routing of the mobile vessels.

1. Scheduling of recipe steps
   Several batches and products are handled in parallel, thus the best production sequence must be decided. A schedule must guarantee that the steps of one product are executed in the sequence defined by its recipe.
2. Assignment of equipment
   Each recipe step must be assigned to a vessel, an AGV and a station. The chosen equipment must be available at the desired time and has to possess the necessary technical functions.
3. Routing of AGVs
   After the assignment, the selected AGV must pick up the vessel and transport it to the selected station. The AGVs must not collide with each other during transports.

The time that a recipe step needs for its execution at a station may depend on the chosen equipment and on the routing of the vessels. A heating step for example takes longer when the material has cooled down while the vessel waited for further processing. The mass and the temperature of the content of a vessel are therefore included as state variables in our simulation model.

1.2 Literature Review

Since the idea of pipeless plants came up in the early 1990s, the scheduling of pipeless plants has been a topic of research. The problem is mostly modeled by mixed-integer linear or nonlinear programming (MILP, MINLP). The mathematical programs are often combined with other methods. [1] and [2] combine the MILP with a discrete event simulator to model the processing durations. The dynamics of the processing durations are neglected in most other publications. [3] use scheduling heuristics and [4] extend their MINLP with a queuing approach. [5] consider constraint satisfaction techniques to find an optimal schedule and deal with vessels that possess their own moving device. Then the assignment of a vessel to an AGV is not necessary. Many publications assume fixed travel times, and a detailed routing of the AGVs and the elimination of collisions are often omitted.

The problem of efficiently operating a pipeless plant has often been simplified in the literature. To our knowledge, only in the modeling, simulation and heuristic scheduling environment PPSIM [6] the problems mentioned in Section 1.1 are considered simultaneously without simplifying assumptions. This PhD work proposes an evolutionary scheduling algorithm to identify the optimal sequencing of the recipe steps that minimizes the makespan. The individuals are evaluated by the simulation algorithm of PPSiM.

2 Research and Study

2.1 Goals

The goal of this PhD research is to enhance the existing modeling and simulation software tool for pipeless plants [6] by an optimization routine. We want to be able to optimize large scale real life problems in a reasonable amount of running time. It was decided to optimize by an EA as it does not rely on explicit information about the problem and its structure but can evaluate the quality of a proposed solution by simulation. The EA should be combined with the simulator in an efficient way. The work to be accomplished can be divided into three work packages:

1. Study the existing software and the related literature.
2. Implement the evolutionary optimization algorithm.
   (a) Decide on the representation of individuals.
   (b) Handle infeasibilities due to the highly constrained problem.
   (c) Integrate the optimization algorithm into the existing software.
3. Analyze and improve the new algorithm.
   (a) Test the algorithm on different problems.
   (b) Tune the parameters.
   (c) Improve the efficiency of the algorithm in terms of running time and performance.

2.2 Current Status

An EA responsible for the scheduling problem in the operation of pipeless plants has been implemented. Repair algorithms aimed at reducing the number of infeasible individuals have been designed and the algorithms were successfully integrated into the simulation software. The task of the simulation software is to provide a heuristic initial solution and to evaluate the individuals by solving the assignment and routing problems by heuristics. The work done so far was published in a research report [7] and was presented on a workshop on computational intelligence in December 2007 [8]. I have furthermore submitted a paper to the European Symposium on Computer Aided Process Engineering [9], which takes place in June 2008, and to a special issue of the Journal of Heuristics [10]. The original ASAP solution of [6] could be improved by adapting the possible starting times of recipe steps. The makespan could be improved by up to 18 %.

Additionally, the computation time was significantly reduced by up to 79 % by specifying the sequence of recipe steps by the EA [10]. I am currently busy with work package 3a. First tests on small problems provide promising results, however, for larger problems there are still many infeasible individuals and their evaluation is time consuming.

2.3 Future Planning

After the tuning of the parameters, the next step is to improve the efficiency of the algorithm. There are three lines of future research.

1. The evaluation of candidate solutions via simulation and heuristics requires much calcuation. It takes between 0.2 and 1.5 seconds to evaluate one individual, depending on the chosen problem example. To speed up the calculations, it is planned to work with approximations in the routing heuristic. The individuals can be evaluated exactly after several generations of the algorithm. This means that there are generations of the algorithm that include possibly infeasible individuals, which implies an uncertainty in the cost function of the EA. I will investigate how the EA can handle the interplay of exact calculations and approximations.
2. Figure 1 illustrates a problem with the assignment of the equipment. Suppose that the first two recipe steps of the shown chromosome have already been scheduled on AGVs. The third recipe step should now be scheduled as soon as possible. It would fit in the gap of the occupation of AGV 2, but then the previously calculated routing of recipe step 1 would be wrong since the AGV would then start at a different location in the plant. In the current implementation only equipment that is not used at a later point in time can thus be chosen, in this example AGV 1. Since this selection may lead to a deterioration of the quality of the solution, the integration of gap filling algorithms will be explored.

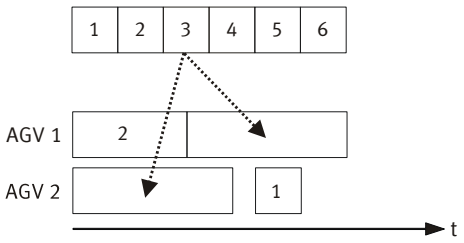

Fig. 1. Gap filling problem

3. The heuristics used by the simulator to evaluate the fitness of the individuals may limit the EA by not allowing sufficient exploration of the available decisions. The assignment of the equipment should therefore be optimized

as well. The equipment can be varied via a second evolutionary layer. Each individual is evaluated by the currently implemented heuristics. Then the new EA searches alternative equipment for a random recipe step. The routing decisions, however, should not be given to an EA. Instead more research will be done to improve the routing heuristic by incorporating possible blockings into the routing decisions [11].

After finishing my PhD I would like to go abroad as I have always liked living in other countries and getting to know different cultures. I could imagine to work as a PostDoc in the area of operations research and optimization. Coming from a mathematical background, I enjoyed applying optimization to problems from chemical engineering. I am now interested in getting to know new challenging areas for optimization. In the long run, I want to take a commercial career, e. g. in a consulting company, to optimize real world problems.

2.4 Study

I am enrolled as a PhD student at the "Graduate School of Production Engineering and Logistics"[1] and receive a scholarship from the Ministry of Innovation, Science, Research and Technology of the State of North Rhine-Westphalia. The graduate school offers interdisciplinary courses, seminars and research projects in the areas of production and logistics. The study comprises three years. Next to their scientific research, the participants of the graduate school have to obtain a total of 30 credits on lectures and exercise courses, on soft skill courses such as moderation and presentation techniques, and on scientific colloquia in which the students regularly present their work and discuss problems with each other. I am in my final year; my scholarship ends in September 2008. I have finished the course program and obtained all required credits such that I can dedicate the remaining time to finishing my PhD research and writing the thesis.

3 An Evolutionary Scheduling and Simulation Approach

The operation of a pipeless plant is a set of complex interacting problems. Making decisions for all problems at once is too time consuming, therefore the problems are treated hierarchically. The EA solves the scheduling problem and the problems of assignment, routing and determining the durations of recipe steps are solved via heuristics and simulation.

3.1 Methodology

The individuals are encoded in an operation-based chromosome as the sequence of the execution of the recipe steps of all batches. The steps are identified by their batch IDs to maintain precedence relations in the recipes during recombination and mutation. The idea is illustrated in Figure 2. The order of the

[1] www.mb.uni-dortmund.de/graduate

sequence indicates the priority of the recipe steps. A chromosome entry directs the schedule builder to schedule the first unscheduled recipe step of the corresponding batch as soon as possible without delaying higher-priority steps. It is an indirect chromosome since a schedule builder is necessary to calculate a full schedule. Since each batch ID must occur a certain number of times in the chromosome, a permutation-based recombination operator is the obvious choice. The order crossover is used here and a swap mutation is performed on the newly generated individuals. The parents are selected in a tournament. We use the $(\mu + \lambda)$ generation strategy with a replacement selection.

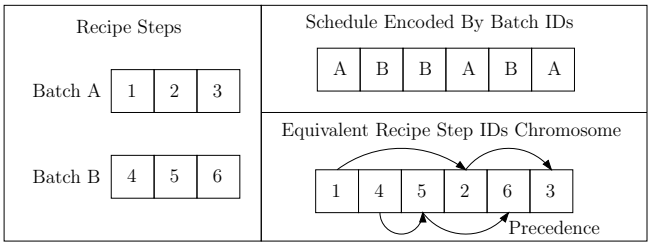

Fig. 2. Encoding of the chromosome

Constraint Handling There are several possible reasons for infeasibility of new individuals. They can be divided into three classes and require different treatment as listed in Table 1. The simplest reason for infeasibility is the violation of the precedence constraints in a recipe. This is avoided by the encoding into an operation-based chromosome. Infeasibilities of class II depend only on the scheduling decisions and can be checked before the simulation run. They are solved by repair algorithms that employ problem-specific knowledge on recipes, vessels and stations. Infeasibilities of class III depend on the output of the simulation. They are only realized after the transfer times and processing times have been calculated and are penalized by the schedule builder after the evaluation of the individual.

	Reason for infeasibility	Treatment	
Class I	Precedence violation	Chromosome	Direct constraint handling
Class II	Occupied stations Too few vessels	Repair	Indirect handling of explicit constraints
Class III	Time constraint violated No route found	Penalty	Indirect handling of implicit constraints

Table 1. Treatment of infeasibility

Repair Algorithms Although repair algorithms have the disadvantage of possibly steering the EA into the wrong direction, the implementation of the two repair algorithms described below is advantageous to reduce the number of infeasible individuals. Performing a repair inside the EA before passing the solution to the schedule builder has shown to provide better results than when dealing with the infeasibility inside the simulator by postponing currently infeasible steps.

The most important property to be satisfied by the repair algorithms is that the meaning of the chromosome, namely to define a list of priorities, is maintained. Recipe steps later in the chromosome should not delay recipe steps from an earlier position in the chromosome. However, if a recipe step cannot be scheduled at its position, its execution may be made possible by inserting a step of lower priority.

It is possible that a newly created individual is infeasible because a recipe step needs to be processed at a station at which another batch is still being executed or waiting for its next transport. The repair algorithm is explained by the example individual in Figure 3. Suppose that there are only two charging stations in the plant. After the fourth recipe step (charging batch 3) both charging stations are occupied by batch 2 and batch 3. The next step (charging batch 4) is thus infeasible. The first recipe step in the sequence that frees a charging station is the seventh entry (heating batch 3). By inserting this step before the currently infeasible step, batch 3 is moved from the charging station to the heating station and batch 4 can be processed. It must, however, be checked that a heating station is available. If not, another repair is needed.

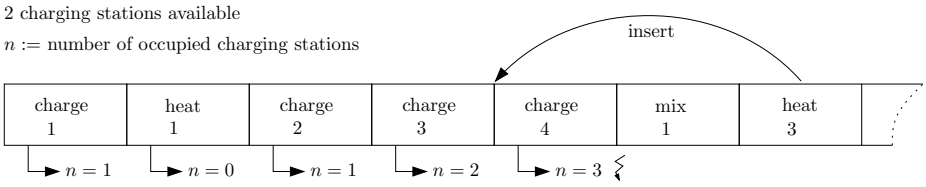

Fig. 3. Repair algorithm for occupied stations

The second repair algorithm ensures that there is a free vessel available when a new batch should be started. The number of vessels is fixed as we assume a given plant topology. The algorithm determines for each chromosome position how many batches are in process. If at some point this number exceeds the number of vessels, all recipe steps of the currently infeasible batch are moved to a position after a batch has been finished and thereby frees a vessel. Note, that entries of the batch which occur behind this position, are not moved forward as this would violate the specified order of the priorities.

Both repair algorithms are executed in an alternating way until the solution is feasible. Since the algorithms may perform conflicting exchanges of recipe steps, it may happen that feasibility is never established. To prevent an infinite loop, the algorithm for repairing an occupied vessel is adapted after some iterations.

Instead of moving (some recipe steps of) a batch to a position where another batch finishes, the complete batch is moved to the end of the chromosome. Thereby it is ensured that the feasibility with respect to occupied stations is maintained and a feasible state is reached.

Simulation Optimization The software PPSiM [6] provides an environment to model a pipeless plant and to perform simulation studies to compare design alternatives. The EA is embedded into PPSiM to perform an optimization of the decision variables. PPSiM computes a production schedule by the simple ASAP heuristic in which the recipe steps are scheduled in a chronological order according to their earliest possible start times. The initial population of the EA consists of multiple copies of this solution and other random solutions. The software is then used as a schedule builder in the evaluation of individuals to map the genotype (a chromosome of batch IDs) to the phenotype (a feasible schedule). It assigns an available and suitable triple of equipment (a vessel, an AGV and a station) with minimum execution time to each recipe step. The execution time consists of the transfer time of the AGV to pick up the vessel, the time needed to transport the vessel to the station and the duration of the recipe step at the station. The transfer times are calculated by the A* algorithm with subsequent collision elimination. Collisions are avoided by waiting and evasion following a first-come-first-serve (FCFS) priority rule. Keeping track of the state changes of the masses, the temperatures and the concentrations of the contents of the vessels and knowing the parameters of the vessels and the stations (heating power, maximum mass flows etc.), the program simulates the exact processing times of the recipe steps at the station. If it is impossible to find a feasible schedule for a candidate solution, the simulator assigns a penalty to the fitness value. It can, for example, happen that AGVs or vessels are blocked by other equipment such that no route can be found for a recipe step. The penalty is the larger the more steps cannot be scheduled. The framework is shown in Figure 4 where it can be seen which tasks are accomplished by the graphical interface of PPSiM, the simulator and the optimizer.

3.2 Experiment

The combination of the EA for the scheduling of the recipe steps and the evaluation by simulation and heuristics is applied to three example problems: a small test problem, an industrial problem and a benchmark problem from the literature. Table 2 summarizes the characteristics of the considered problems. The exact recipes and the plant layouts are available from the author upon request. The strategies used for the different modules of the EA as well as the parameter settings are summarized in Table 3. We perform 50 independent runs.

Example 1 This is a simple example since neither tight timing constraints such as different starting times or deadlines nor much potential for routing conflicts are present. In Example 1a the AGVs drive at a speed of 0.025 m/s while the speed in Example 1b is reduced to 0.01 m/s.

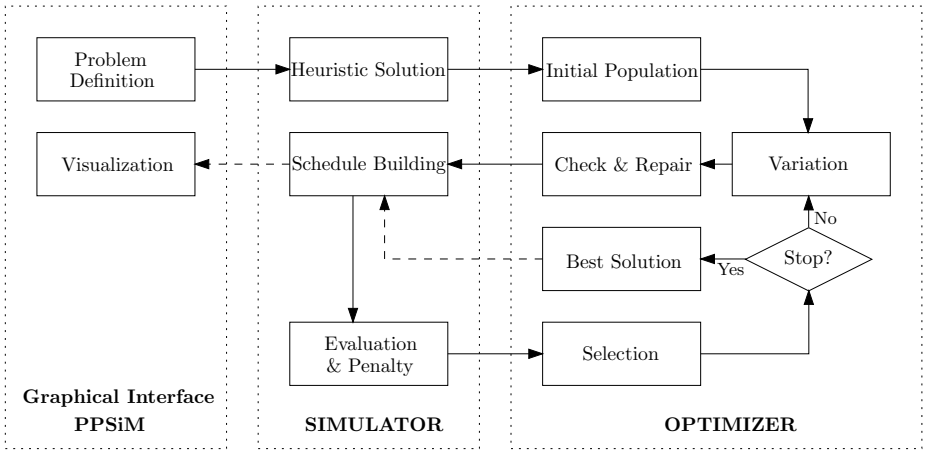

Fig. 4. Simulation optimization for pipeless plants

	# Recipe steps	# Vessels	# Stations	# AGVs	# Batches
Example 1	18	3	6	2	4
Example 2	84	10	7	2	7
Example 3	35	3	8	3	5

Table 2. Characteristics of the examples

Example 2 The underlying production process is a mixing process in which various substances are charged and the material is mixed, heated and cooled. The plant area is divided into a production area, an intermediate storage and parking area, a discharging area and a cleaning area. The production stations are arranged in a clockwise order to minimize the probability of collisions or interlockings during the execution of the recipes. The stations are cleaned in place for 10 minutes after each usage.

Example 3 A herringbone layout (Example 3a) and a linear layout (Example 3b) for a problem from the literature [5] are considered. The recipe steps have fixed processing durations and the AGVs move on specified paths of fixed transfer times. Thus, a detailed simulation of the chemical processing steps and of the routing of the AGVs is unnecessary. The vessels possess their own moving devices, which renders the assignment of an AGV to a vessel unnecessary. There are buffers such that one AGV can wait in front of a station and one behind a station. Furthermore there are buffers at certain locations in the plant at which AGVs can pass each other. This simplifies the routing problem significantly and it is possible to use the buffers at the stations as a parking position for empty and clean vessels. We implement the example without buffers and define additional parking positions at the side of the plant layout. To be able to compare the

objective	minimum makespan
initialization	ASAP heuristic and random initialization
representation	operation-based chromosome
population size	$\mu = 10$
parent selection	tournament of size 5
recombination	order crossover
recombination probability	100 %
mutation	swap
mutation probability	80 %
offspring	$\lambda = 2$
survival selection	$(\mu + \lambda)$ replacement selection
termination criterion	60 seconds

Table 3. Modules and parameters of the EA

results we deduct the times of the additional transfers to these parking positions from the makespan.

3.3 Results

The results obtained by the presented EA with repair algorithms and evaluation by simulation and heuristics are shown in Table 4. The percentages indicate the improvement with respect to the initial ASAP solution. The last column shows the standard deviation of the found solutions. The improvement for Example 1 is larger than for the industrial Example 2. The best solutions for Example 1 produce the batches not in parallel but rather in a sequential mode. It seems important not to run too many batches simultaneously. It is a weakness of the ASAP solution that it starts many batches at once. Thereby vessels are blocking each other. For the industrial problem, however, it seems that the ASAP solution has already provided a very good solution. We suppose that this is due to the efficient plant layout which minimizes the possibility of collisions.

	best		median		worst		σ
1a	4652	(-12.5 %)	4676	(-12.0 %)	4700	(-11.6 %)	15
1b	6176	(-17.1 %)	6249	(-16.2 %)	6303	(-15.4 %)	27
2	82751	(-2.6 %)	84780	(-0.2 %)	84929	(-0.0 %)	522
3a	44444	(-4.8 %)	44561	(-4.5 %)	44720	(-4.2 %)	78
3b	47320	(-14.2 %)	50781	(-7.9 %)	51880	(-6.0 %)	1788

Table 4. Results of the EA in comparison to the ASAP solution

For a comparison with the results obtained by constraint satisfaction [5] on Example 3, Table 5 shows the first found solution and the best found solution for both layouts. In our case the first solution is the ASAP solution. Our best

	EA		Constraint Satisfaction [5]	
	CPU time	makespan	CPU time	makespan
Herringbone (Example 3a)				
First solution	1.97	46680	0.1	63720
Best solution	25.5	44444	722.6	44280
Linear (Example 3b)				
First solution	2.31	55161	0.12	70200
Best solution	31.5	47320	876.44	48600

Table 5. Comparison for Example 3

solution for the linear layout is better than the supposedly optimal solution obtained with constraint satisfaction techniques. Since the authors of [5] did not explicitly state whether the transfer times are the same in both layouts, we suspect that we are working with different transfer times in the linear layout. The CPU times are given in seconds, however, they cannot be compared with each other since the CPU performances are different. We use an Intel Pentium 4 with 3 GHz and 2 GB RAM, which has a higher computing power than the P3500 MHz PC of [5]. The results show that the heuristic ASAP solution is found later than the first solution found by constraint satisfaction, but that it has a significantly smaller makespan. The EA improves this solution to a solution very close to the optimum. For both layouts the best solution was found before the time limit of one minute. The progress plots of the EA are shown in Figures 5 and 6. It can be seen that always one of the randomly created solutions of the initial population outperformed the ASAP solution.

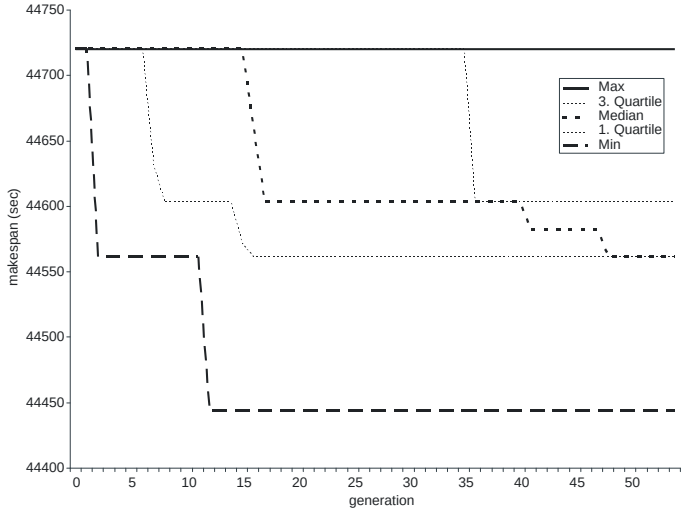

Fig. 5. Progress plot for Example 3a

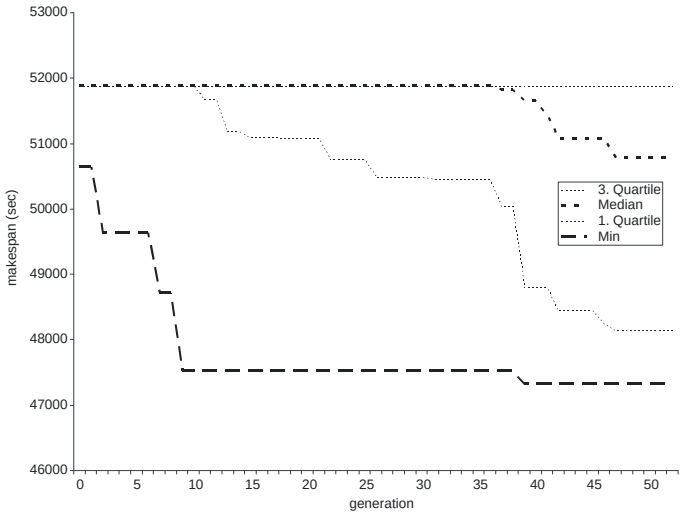

Fig. 6. Progress plot for Example 3b

4 Achievements

As with many real-world problems, the optimization of the operation of a pipeless plant is too complex to be solved by an exact approach. Therefore it is treated by simulation optimization in a hierarchical way. An EA schedules the production steps and a simulator with heuristics for the assignment of the equipment and for the routing of the vessels is used as the schedule builder. In examples, the new scheduling procedure decreased the makespan by up to 17 % compared to the initial ASAP solution. For a problem from the literature we found a good first solution and the EA quickly returned a solution very close to the optimum. In my final thesis, I hope to have developed an efficient algorithm to optimize the operation of a pipeless plant taking into account the detailed routing and conflict elimination and the variable processing times that have not been treated explicitly in the literature yet.

5 Feedback

I highly appreciate getting the chance to present my PhD work in the Doctoral Consortium as I hope to receive useful feedback and hints from experts on the topic of evolutionary computing. I am looking forward to fruitful scientific discussions and I am keen on getting to know about other complex real world problems that have been successfully tackled with EAs. I would like to know whether there are other applications in which scheduling, assignment and routing problems interact in such a dynamic way as it is the case in the operation of pipeless

plants. Any suggestions, ideas for improvement and critical questions are very welcomed. Particularly interesting are the topics on how to efficiently integrate repair algorithms into the EA and how to deal with uncertainties when working with approximations. I consider the international composition of the Doctoral Consortium a very interesting opportunity for networking and exchanging of ideas.

6 Acknowledgements

I am grateful for the support of the "Graduate School of Production Engineering and Logistics" and I would like to thank the anonymous reviewers for their helpful comments and suggestions. Their advices are highly appreciated.

References

1. Gonzalez, R., Realff, M.: Operation of pipeless batch plants - I. MILP schedules. Computers & Chemical Engineering **22**(7 - 8) (1998) 841 – 855
2. Gonzalez, R., Realff, M.: Operation of pipeless batch plants - II. Vessel dispatch rules. Computers & Chemical Engineering **22**(7 - 8) (1998) 857 – 866
3. Panek, S., Engell, S., Lessner, C.: Scheduling of a pipeless multi-product batch plant using mixed-integer programming combined with heuristics. In: Proceedings to the 15th European Symposium on Computer Aided Process Engineering. (2005) 1038 – 1088
4. Yoo, D., Lee, I.B., Jung, J.: Design of Pipeless Chemical Batch Plants with Queueing Networks. Industrial & Engineering Chemistry Research **44** (2005) 5630 – 5644
5. Huang, W., Chung, P.: Integrating routing and scheduling for pipeless plants in different layouts. Computers & Chemical Engineering **29** (2005) 1069 – 1081
6. Liefeldt, A., Engell, S.: A Modelling and Simulation Environment for Pipeless Plants. In: Proceedings to the 8th International Symposium on Process Systems Engineering. Volume 15B., Elsevier, B. Chen and A.W. Westerberg (2003) 956 – 961
7. Engell, S., Piana, S.: Scheduling of Pipeless Plants using Evolutionary Algorithms. Research Report, NRW Graduate School of Production Engineering and Logistics (November 2007)
8. Piana, S., Engell, S.: Evolutionäre Optimierung des Betriebs von rohrlosen Chemieanlagen. In: Proceedings 17. Workshop Computational Intelligence, Universitätsverlag Karlsruhe (2007)
9. Piana, S., Engell, S.: Computer Aided Operation of Pipeless Plants. Accepted for European Symposium on Computer Aided Process Engineering ESCAPE 18 (2008)
10. Piana, S., Engell, S.: Simulation Based Evolutionary Optimization of the Operation of Pipeless Plants. Submitted to Journal of Heuristics Special Issue on Advances in Metaheuristics
11. Möhring, R., Köhler, E., Gawrilow, E., Stenzel, B.: Conflict-free Real-time AGV Routing. In Fleuren, H., den Hertog, D., Kort, P., eds.: Operations Research Proceedings, Springer (September 2004) 18 – 24