Reducing the Number of Fitness Evaluations in Graph Genetic Programming Using a Canonical Graph Indexed Database

Jens Niehaus niehaus@vsys.de
Visual Systems Automation GmbH, 59174 Kamen, Germany

Christian Igel igel@neuroinformatik.rub.de
Institut für Neuroinformatik, Ruhr-Universität Bochum, 44780 Bochum, Germany

Wolfgang Banzhaf banzhaf@cs.mun.ca
Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, A1B 3X5, Canada

Abstract

In this paper we describe the genetic programming system GGP operating on graphs and introduce the notion of graph isomorphisms to explain how they influence the dynamics of GP. It is shown empirically how fitness databases can improve the performance of GP and how mapping graphs to a canonical form can increase these improvements by saving considerable evaluation time.

Keywords

graph representation, genetic programming, graph isomorphism, neutrality, fitness database.

1 Introduction

In genetic programming (GP) (Koza, 1994; Banzhaf et al., 1998), the elements of the search space are computer programs. These programs are expressions of a predefined formal language. The genotype of an individual traditionally corresponds to the parse tree of the expression it represents. In other GP systems, genotypes may encode not only trees, but also more general graph structures.

If a search space consists of trees or general graphs, graph isomorphisms influence the dynamics of the optimization algorithm, because isomorphic graphs (under some restrictions) encode functionally equivalent solutions. In terms of evolutionary computation, isomorphisms induce neutral encodings, that is, non-injective genotype-phenotype mappings. The effects of graph isomorphisms on search were investigated in the context of optimization of Bayesian networks (graphical models), where it was argued that it is often more appropriate to search among classes of equivalent—i.e., isomorphic—networks than among individual network topologies (Chickering, 2002). In the domain of architecture optimization of neural networks, it was shown that graph isomorphisms can lead to a biased representation of phenotypes (Stagge and Igel, 2000, 2001) and can considerably influence search performance (Igel and Stagge, 2002a). It was further demonstrated that a database which stores fitness evaluations to speed up optimization can save a significant amount of evaluations if considered together with isomorphisms (Igel and Stagge, 2002b).

©2007 by the Massachusetts Institute of Technology

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

Here, we present a GP system called GGP introduced by Niehaus and Banzhaf (2001) that operates on graphs. We investigate the potential of accelerating GP with the help of a fitness database that takes isomorphisms into account. Our general considerations are independent of the GP implementation, yet are supported here by experiments with GGP.

The article is organized as follows. In Section 2, we describe the GGP algorithm and introduce the concept of dynamic demes. In Section 3, GGP is applied to three common GP benchmark problems, usually tackled by tree-representations. The same benchmark problems are later used to study the effects of graph isomorphisms. Optimization results are compared to those of other GP systems to verify that using GGP for these tasks is appropriate. In particular, we study the use of dynamic demes. Then, in Section 4, we present some results from graph theory that help to analyze and handle phenomena caused by isomorphisms in evolutionary algorithms. Section 5 applies those methods within the context of GGP to demonstrate the effects of isomorphism in practice. A canonical graph indexed fitness database is used for reducing the number of fitness evaluations. In this context the relation of isomorphisms and phenotypic neutrality in GP is established and discussed.

2 Graph-based Genetic Programming

In the following, we sketch the graph-based GP system GGP. There are several reasons for using GGP in this investigation. First, because our theoretical concepts should not be limited to trees, we want to study them in the context of a "general" graph-based GP system. Second, we want to introduce our graph-based algorithm as an alternative GP system. Third, we want to demonstrate a key advantage of GGP, namely that graphs are represented "directly", i.e., without a complex mapping from genotype space to graph search space. This makes the results much more accessible in the representation discussed here.

2.1 Outline of GGP

The idea to automatically design automata, whose structures are ultimately graphs, dates back to the middle of the last century (Turing, 1950; Fogel, Owens, and Walsh, 1966). In many GP systems the genotypes are trees that represent expressions of a formal language, originally *LISP* functions (Cramer, 1985; Koza, 1992). Recent GP implementations consider other data structures, among them graphs with *graph-encoding* and *graph-based* approaches distinguishable. Graph-encoding GP arose from the need to evolve graph structures (e.g., electric circuits). Although the phenotypes are graphs, the genotypes in graph-encoding GP systems may still be trees as in the work of Koza et al. (1999). Teller and Veloso (1995) were the first to use general graphs as genotypes for algorithmic reasons—independent of whether the phenotypes have general graph-structure or not. We refer to such implementations as *graph-based* GP systems. More recently, hybrid systems have been implemented (Kantschik and Banzhaf, 2002).

Our GP system GGP, first introduced by Niehaus and Banzhaf (2001), is both graph-based and graph-encoding. A detailed specification of GGP is beyond the scope of this paper, we refer to the descriptions by Niehaus and Banzhaf (2001) and Niehaus (2003). In the following, we describe the graph encoding used in GGP, variation operators used, and finally the steady-state reproduction and selection method and the *dynamic demes* concept.

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

2.2 Representation

GGP uses colored (i.e., labeled) directed multigraphs for the internal representation of candidate solutions. A multigraph is a graph where there can be several edges between vertices (therefore connections E are described by multisets, denoted by $\{\!\{\ldots\}\!\}$). Every node has a label or color (the terms color and label are used synonymously throughout this article). Each color represents one of the operations given in the problem definition, such as $+$, x_1, 1, or `move`. If not stated otherwise, we presume the canonical semantics of labels in GP expressions. In-degree and out-degree of all vertices with the same color must be identical. This ensures that functions which require a certain fan-in or fan-out always have the correct number of input and output connections. For operations that could handle a variable number of arguments, distinct functions for each arity have to be introduced (e.g., $+_1$ and $+_2$ for adding two or three numbers, respectively).

The vertices of a graph in GGP can be divided into three sets V_{in}, V_{inner}, and V_{out}. The set V_{in} consists of the input vertices, which represent the input values of the problem at hand and the starting points for parsing the graph. Similarly, the outputs are represented by output vertices in V_{out}. Each vertex in $V_{\text{in}} \cup V_{\text{out}}$ has its own unique color. Input vertices have an in-degree of 0 and an out-degree of 1, output vertices vice versa. We denote a vertex with an in-degree of n and an out-degree of m as an (n, m)-*vertex*. The set V_{inner} is a variable-sized set of inner vertices corresponding to elements of the function set of the problem. Formally, a feasible *GP-graph* in the GGP system can be defined as follows.

Definition 1. *A directed multigraph $\tilde{G} = (V_{in} \cup V_{inner} \cup V_{out}, \tilde{E})$ is called a GP-graph, if and only if*

1. *$V_{in} = \{i_1, \ldots, i_n\}$ and $V_{out} = \{o_1, \ldots, o_m\}$ are sets of n and m ordered input and output vertices, respectively;*
2. *$V_{inner} = \{v_1, \ldots, v_l\}$ is a finite set of l inner vertices;*
3. *$\tilde{E} = \{\!\{(a, b) \mid a \in V_{in} \cup V_{inner} \wedge b \in V_{inner} \cup V_{out}\}\!\}$ is a multiset of edges;*
4. *Each input vertex $i \in V_{in}$ is a $(0, 1)$-vertex;*
5. *Each output vertex $o \in V_{out}$ is a $(1, 0)$-vertex;*
6. *For each $v \in V_{inner}$ there is an order defined on both multisets of edges $\{\!\{(a, v) \mid a \in V_{in} \cup V_{inner} \wedge (a, v) \in \tilde{E}\}\!\}$ and $\{\!\{(v, b) \mid b \in V_{inner} \cup V_{out} \wedge (v, b) \in \tilde{E}\}\!\}$;*
7. *For each $v \in V_{in} \cup V_{inner}$ there exists a path connecting v with an element $o \in V_{out}$;*
8. *For each $v \in V_{inner} \cup V_{out}$ there exists a path from an input vertex $i \in V_{in}$ to v;*
9. *If the search space consists of acyclic graphs only, the directed multigraph must also be acyclic.*

While $V_{\text{in}} \cup V_{\text{out}}$ is fixed for all individuals throughout the evolutionary process, V_{inner} and $\tilde{E}$ are subject to adaptation. The edges are a multiset, because a node can get more than one input from another node. Ordered edges (item 6) are needed because they allow us to use the vertices as representations for non-commutative functions (e.g., to distinguish between $(-\,x_1\,2)$ and $(-\,2\,x_1)$). The 7th item ensures that each node in a GP-graph is connected to an output. Thereby, the occurrence of introns is reduced and the graphs are kept small to prevent bloat. When cyclic graphs are evolved, item 8 is

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

needed to ensure that the genetic program represented by the graph can be evaluated. It ensures that each inner node gets input from at least one $(0,1)$-vertex.

So far, nodes representing different functions cannot be distinguished in a GP-graph. This is achieved by assigning a coloring to each GP-graph. Let there be k different types (color classes) of nodes and let π be a function assigning nodes of a graph to their color class (see Section 4 for a more formal definition), that is, for all $v_1, v_2 \in V_{\text{in}} \cup V_{\text{inner}} \cup V_{\text{out}}$ it holds $\pi(v_1) = \pi(v_2)$ if and only if v_1 and v_2 have the same color.

Definition 2. *The coloring π of the GP-graph $\tilde{G} = (V_{in} \cup V_{inner} \cup V_{out}, \tilde{E})$ is feasible if and only if for all $v_1, v_2 \in V_{in} \cup V_{inner} \cup V_{out}$ and $v_1 \neq v_2$*

1. *If $v_1 \in V_{in} \cup V_{out}$ then $\pi(v_1) \neq \pi(v_2)$.*
2. *If v_1 is a (n,m)-vertex and $\pi(v_1) = \pi(v_2)$ then v_2 is a (n,m)-vertex.*

The first item formalizes the condition that each input or output node has a unique label (e.g., in order to distinguish between different inputs). The second item enforces the condition that in-degree and out-degree of all equally colored vertices must be identical. In the following, we consider only feasible colorings.

2.3 Operators

Special crossover and mutation operators are needed for traversing the search space of GP-graphs. We only consider operators that generate feasible GP-graphs from GP-graphs, therefore no repair mechanism is needed. Whenever a new vertex is inserted, its color is chosen randomly from all possible colors with compatible input and output degree.

First, we discuss mutations that operate on single, randomly chosen nodes:

Vertex mutation (*vmut*): The operator selects one vertex $v \in V_{\text{inner}}$ and changes its function (color) to one with the same in- and out-degree. The order of the edges remains the same.

Vertex deletion (*vdel*): A $(1,1)$-vertex v is chosen, deleted, and the adjacent edges (u,v) and (v,w) are replaced by a (u,w).

Vertex insertion(*vins*): An edge $(u,w) \in E$ is split and a $(1,1)$-vertex v is inserted resulting in two new edges (u,v) and (v,w).

Vertex movement (*vmov*): A $(1,1)$-vertex connected to vertices u and w is replaced by an edge (u,w) and inserted differently afterwards such that it is still connected to either u or w. An example for the *vmov* variation is given in Figure 1.

It is not possible to simply remove a (n,m)-vertex with $n \neq m$ because this would lead to infeasible graphs. Thus, we use operators capable of inserting / deleting subgraphs. We define *GP-paths*, which are closely related to the ordinary notion of paths in graphs:

Definition 3. *A subgraph $(\hat{V}, \hat{E})$ of a GP-graph $(V_{in} \cup V_{inner} \cup V_{out}, \tilde{E})$ is called GP-path, if and only if*

1. *$\hat{V} = \{v_0, v_1, \ldots v_n\} \subset V_{inner}$ and $\hat{E} = \{e_1, \ldots, e_n\} \subset E$;*
2. *$e_i = (v_{i-1}, v_i)$ for each edge $e_i \in \hat{E}$;*
3. *The vertices $v_1, \ldots, v_{n-1}$ are $(1,1)$-vertices, the vertices v_0 and v_n are not;*

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

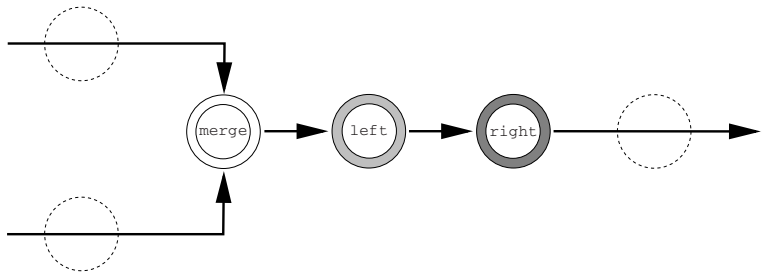

Figure 1: Example for the vertex movement variation *vmov*: When *vmov* is applied to the middle node with label `left`, the node is deleted and randomly inserted at one of the three positions marked by dashed circles.

4. *v_0 has an out-degree higher than one or is a $(0,1)$-vertex, v_n has an in-degree higher than one.*

Figure 2 shows a GP-graph with corresponding GP-paths. The following operators are designed to insert and delete GP-paths.

GP-path deletion (*pdel*): This operator deletes a GP-path $\{v_0, \ldots, v_n\}$. If v_0 is a $(0,1)$-vertex, it is directly removed. Otherwise, the (a,b)-vertex v_0 is replaced by a $(a, b-1)$-vertex with an appropriate color (a $(1,2)$-vertex can additionally be replaced just by an edge). The (c,d)-vertex v_n is replaced by a $(c-1,d)$-vertex with conforming color. The vertices $v_1, \ldots v_{n-1}$ and the set of edges $\hat{E}$ are deleted. The implementation of the operator ensures that only GP-paths are chosen that can be removed without violating the constraints in definition 1.

GP-path insertion (*pins*): This reverses the operation of deleting a GP-path. The newly inserted GP-path always consists of two vertices. The operator obeys requirement 9 of definition 1.

Cycle creation (*cycle*): This operator is a variation of *GP-path insertion*. It ensures that the new GP-path creates a cycle in the graph. This operator must not be applied for problems where only acyclic graphs are feasible.

Crossover (*xover*): The most complex operator creates a GP-graph by combining two graphs. From both graphs a subgraph is extracted. One of those subgraphs is inserted into the remaining part of the other GP-graph. The subgraphs must have the same number of inward directed and outward directed edges (the edges that connect a vertex outside of the subgraph with one inside and vice versa) so that the edges that connected the subgraph with the remaining part of the GP-graph can be replaced by new ones. Hence, the new graph obeys requirements 1 to 5 of the GP-graph definition. Item 6 is fulfilled because the order of the edges of source graphs is preserved. More difficult to fulfill for crossover are items 7 to 9. Currently we have no constructive algorithm ensuring these properties. Instead we check whether the resulting offspring is feasible. If not, crossover is repeated.

2.4 Steady-State Evolutionary Algorithm

The GGP system is basically a steady-state evolutionary algorithm (Whitley, 1989; Syswerda, 1991) with tournament-selection. The initial population consists of n_{pop} ran-

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

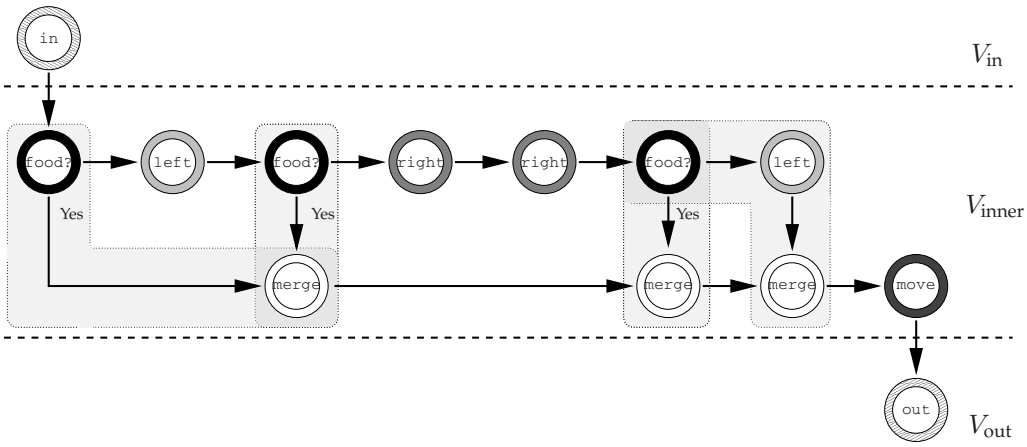

Figure 2: Example of a GP-graph according to definition 1 representing a solution for the Artificial-Ant problem (see Section 3). The boxes highlight all GP-paths, see definition 3.

dom GP-graphs. In every generation, n_{tour} individuals are selected for a tournament. They are evaluated and ranked according to their fitness. In the experiments in this study, we use the size of the individuals as second level sorting criterion. If two individuals have the same fitness, the smaller one is ranked better. This optional feature induces a bias towards graphs with few nodes. Thereafter, variation operators are applied to the n_{winner} best individuals of the tournament and the new resulting individuals overwrite the n_{winner} worst individuals of the tournament. In the case of crossover, the winning individual is recombined with the loosing individual to replace it.

An individual is modified by application of between one and three operators. Empirical studies have shown that there is a strong causality between the number of applied operators and fitness change (Niehaus, 2003). Unfortunately, the fraction of fitness improvements decreases very quickly. We found that one to three variation operations are reasonable to achieve good results. The selection probabilities for each operator and the number of operators applied to create an offspring are free parameters of the GP-System. In this study, these parameters are kept constant. An adaptive algorithm is presented in Niehaus and Banzhaf (2001).

In certain situations some operators may not be applicable at all. Should, for example, the maximum size of the graphs be limited an insertion-operator cannot be applied if the graph already consists of the maximum number of nodes. In such a case, the selection probability for this operator is set to zero.

2.5 Dynamic Demes

The GGP system implements the concept of demes (D'haeseleer and Bluming, 1994). Instead of fixed deme sizes with a constant migration rate we use our own method called *dynamic demes* (Niehaus, 2003): If all individuals in a population are descendants of the same individual we split the population into two equally sized demes, keep the better half of the individuals in one and replace all GP-graphs in the second deme with randomly created ones. Demes are split until they reach a fixed minimum size of n_{deme} individuals. Two demes are merged if the fitness values of their best individuals are on the same level. There is no migration of single individuals between different

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

demes. Hence, no individual of an advanced deme can take over another deme that is exploring a different part of the search space but has not achieved comparable fitness values yet.

In order to demonstrate the effects of dynamic demes, we perform all experiments with and without demes in this study.

3 Experimental Evaluation

In this section, the performance of GGP is evaluated on three common GP benchmark problems and compared to results from the literature. We compare GGP with and without dynamic demes. These test problems will also be used for evaluating the graph indexed database in Section 5, hence we need to show that it is indeed reasonable to use GGP to tackle these problems.

3.1 Test Problems

We use three well known benchmark problems in order to assess the performance of GGP. We will confine the problem description to GGP parameters used and to the differences of approaches that arise from using a graph-based GP system. Detailed descriptions of these problems can be found in the cited literature.

The aim of the *Two-Boxes* symbolic regression problem is to find a formula for the difference between the volumes of two boxes (Koza, 1994). The *Lawn-Mower* benchmark is the problem of finding an algorithm for mowing a lawn consisting of $n \times n$ squares. The third benchmark is the Artificial-Ant problem on the *Santa Fé trail* (Koza, 1992), in which an agent has to find 89 pieces of food on a field of 32×32 squares.

Function	Degree
`left`	(1,1)
`right`	(1,1)
`move`	(1,1)
`food?`	(1,2)
`merge`	(2,1)

(a) Artificial-Ant

Function	Degree
`left`	(1,1)
`mow`	(1,1)
`loop`	(1,2)
`merge`	(2,1)

(b) Lawn-Mower

Function	Degree
`add`	(2,1)
`sub`	(2,1)
`mul`	(2,1)
`div`	(2,1)
`nop`	(1,1)
`branch`	(1,2)

(c) Two-Boxes

Table 1: Function sets of the benchmark problems. The second column gives the in- and out-degree of the corresponding functions.

Table 1 shows the function sets used in the benchmark applications. Both the Artificial-Ant and the Lawn-Mower problem have one input and one output vertex. The Two-Boxes problem requires six input vertices (representing width, height, and length of the boxes) and one output vertex. The `loop` function for Lawn-Mower is defined as *continue execution with the vertex adjacent to the first outward directed edge as long as there are movements left*. The functions `merge`, `branch`, and `nop` do not change any variables. The function `merge` is needed because in-degree and out-degree of all vertices representing the same function are fixed. For example in Figure 2, the final `move` preceding the output cannot get more than a single input as it is a $(1,1)$-vertex. To connect more than one (in this case 4) vertices indirectly with the final `move`, `merge` nodes are needed. In the Lawn-Mower and Artificial-Ant tasks the maximum number of movements per trial is 100 and 400, respectively.

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

Problem	*vmut*	*vdel*	*vins*	*vmov*	*pdel*	*pins*	*cycle*	*xover*	no op.
Artificial-Ant	12 %	12 %	12 %	12 %	12 %	12 %	12 %	12 %	4 %
Lawn-Mower	12 %	12 %	12 %	12 %	12 %	12 %	12 %	12 %	4 %
Two-Boxes	24 %	2 %	0	0	24 %	24 %	0	24 %	4 %

Table 2: Operator probabilities used in the experiments. The probability of performing no variation (no op.) is 4 % in all experiments. For Artificial-Ant and Lawn-Mower the GGP system is allowed to create and modify cyclic graphs while it is restricted to acyclic graphs in the case of Two-Boxes.

Table 2 shows the operator probabilities used. The `nop` operation is the only function associated with a $(1, 1)$-vertex in the Two-Boxes task and it is only needed by the random graph generator during the initialization of the population (cf. Niehaus, 2003). Thus, probabilities of vertex-operators are low. Furthermore, the Two-Boxes benchmark is the only problem for which we require acyclic graphs. Hence, the probability of applying the operator `cycle` is set to zero and `xover` is configured not to create cycles. The remaining parameters are given in Figure 3. Note that going from a tree to a graph representation has changed the function set only as far as the switch from a functional to an imperative algorithmic approach is concerned.

Parameter	Artificial-Ant	Lawn-Mower	Two-Boxes
Tournament size n_{tour}	4	4	4
Winner in a tournament n_{winner}	2	2	2
Max. no. of vertices in V	25	40	20
Population size n_{pop}	100	100	100
Experiments	500	500	500
Fitness evaluations n_{eval}	$2 \cdot 10^5$	$2 \cdot 10^5$	$2 \cdot 10^5$
Min. deme size n_{deme}	8	8	8

Table 3: Parameter sets in the experiments.

Results of 500 independent trials per test problem are summarized in Figure 3 and Table 4. The table shows the number of trials in which an optimal solution was found. Further, the average number of fitness evaluations to find a solution is given. The diagram shows the number of trials (out of 500) that found an optimal solution as a function of the number of fitness evaluations.

3.2 Results

The results demonstrate that none of the benchmarks is difficult for graph GP. Especially the results for the Artificial-Ant problem are remarkable, see Table 5, since basic GP algorithms are not much better than random search on this task (Langdon and Poli, 1998). Our own experiments with a tree-based GP system show success rates of only seven percent (Niehaus, 2003). In more recent publications fitness cases are altered from complete paths to chunks of a path resulting in a chance for tree GP to solve the problem (Kuscu, 1998; Langdon, 1998). Here we propose a GP system capable of reliably solving the original problem. An analysis of the best final individual of each trial shows that almost all programs followed the same basic strategy as the solution depicted in Figure 2. That is, GGP evolved small solutions that systematically look for

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

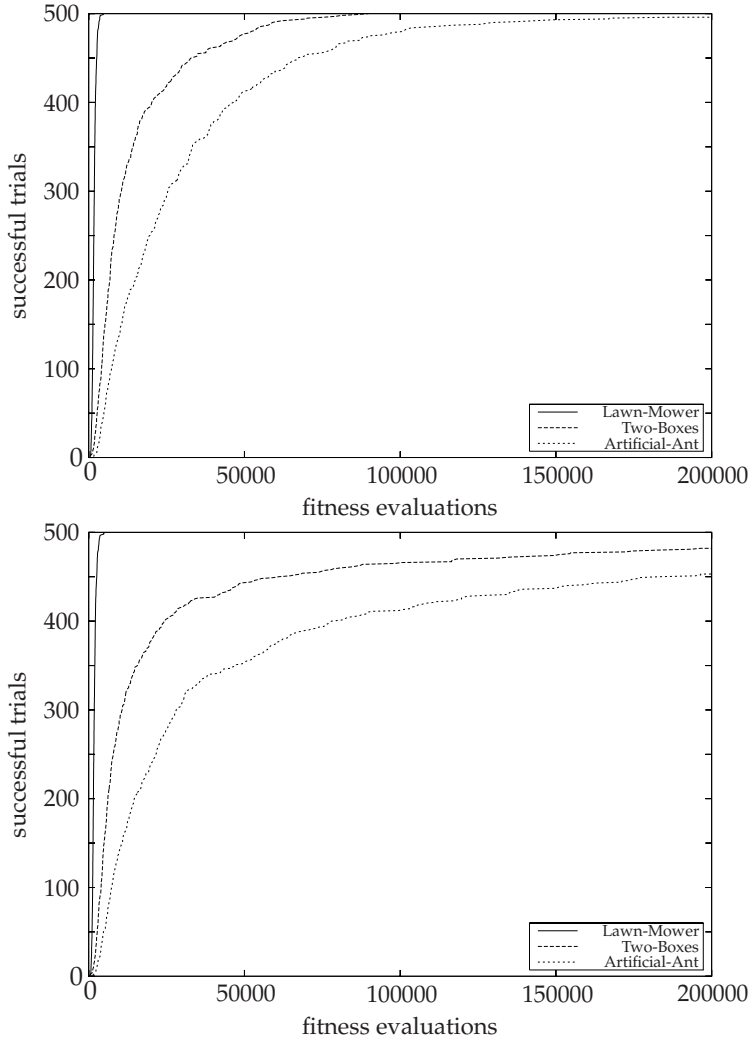

Figure 3: Performance of GGP on the three benchmark problems. The upper plot refers to GGP using demes, the lower plot to GGP without demes. For each problem 500 trials were carried out. The curves show the number of trials that found an optimal solution given the number of fitness evaluations.

food in the neighborhood; it did not learn the pattern of the Santa Fé trail.

The Lawn-Mower benchmark turns out to be especially easy for GGP, see Figure 3 and Table 4. On average, only 1727.9 fitness evaluations were needed to find an optimal solution. In Table 6 we compare these figures to results from the literature. Because in the cited articles often only the computational effort is published, we also use this measure although we are aware of its limitations (Luke and Panait, 2002; Christensen and Oppacher, 2002; Niehaus and Banzhaf, 2003). The results for the Two-Boxes benchmark are given in Table 7. When looking at the final best individuals, it turns out that GGP always found the smallest possible exact solution. As we consider the size of the individuals as second level sorting criterion, small solutions were preferred in our ex-

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

Demes	Lawn-Mower	Two-Boxes	Artificial-Ant
successful trials	500	500	496
avg. evals	1727.9	13510.5	29054.1

No demes	Lawn-Mower	Two-Boxes	Artificial-Ant
successful trials	500	482	453
avg. evals	1743.5	21485.1	37591

Table 4: Results of GGP after $2 \cdot 10^5$ evaluations with and without demes. The number of successful trials out of 500 and the average number of evaluations after which an optimal solution was found are given.

Method	Solutions no.	Solutions fraction	Trials
Original Tree GP (Koza, 1992)	35	17.5%	200
Strict Hill Climbing (Langdon and Poli, 1998)	8	16%	50
Evolutionary Programming (Chellapilla, 1997)	47	79.7%	59
Scalar speed (Langdon and Poli, 1998)	9	18%	50
1 food ahead (Langdon and Poli, 1998)	19	38%	50
5 food ahead (Langdon and Poli, 1998)	71	35.5%	200
1-point Crossover, orig. fitness function (Langdon and Poli, 1998)	8	8%	100
Grammatical Evolution, $n_{\text{eval}} = 10^5$ (O'Neill et al., 2003)	56	56%	100
GGP, without demes	453	90.6%	500
GGP, with demes	496	99.2%	500

Table 5: Results for the Artificial-Ant benchmark. Given are the overall number of trials and the number (no.) and percentage (fraction) of successful ones.

periments (because if two individuals have the same fitness, the smaller one is ranked better). This is usually desired, but there might be problems where this strategy could lead to premature convergence to solutions that are not sufficiently complex.

The performance of GGP benefits from the higher diversity introduced by demes. In experiments without dynamic demes, the success rates for Artificial-Ant and Two-Boxes drop significantly to 91 % and 96.4 %, respectively (Fisher's exact test, $p < .001$). For the Lawn-Mower benchmark, still 100 % of the trials were successful.

From the limited set of toy benchmark problems, one cannot conclude that GGP is in general superior to standard GP for certain classes of applications (unfortunately, even the definition of such problem classes is difficult in GP). Still, the results show that GGP is a competitive alternative to other GP systems and therefore a reasonable testbed for the experiments below. We think that GGP is particularly well suited for problems where the solution can be described by a short, iterative algorithm.

4 Graph Isomorphisms

In this section, we review some basic concepts from graph theory for dealing with the phenomena caused by graph isomorphisms in GP (cf. McKay 1981, 1990).

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

Method	Effort
tree (Poli, 1997)	100000
stack based (Bruce, 1997)	58000
tree + ADF (Poli, 1997)	11000
tree + culture (Spector and Luke, 1996)	10000
parallel distributed (Poli, 1997)	5000
GGP, without demes	3398
GGP, with demes	3091

Table 6: Computational effort (Koza, 1992) for the Lawn-Mower task.

Method	Solutions
Heywood and Zincir-Heywood, 2002	4-36 %
Terrio and Heywood, 2002	4-92 %
GGP, without demes	96.4 %
GGP, with demes	100 %

Table 7: Results for the Two-Boxes benchmark. Success rate ranges are given, since Heywood and Zincir-Heywood (2002) and Terrio and Heywood (2002) tested several approaches with different parameters.

4.1 Colored Graphs

Let $G = (V, E)$ be a directed graph with vertex set V and edge set $E \subseteq V \times V$. We define a *coloring* π with k colors as an ordered collection $(V_1, V_2, \ldots, V_k)$ of k disjoint subsets of V with $\bigcup_i V_i = V$. We do not require $V_i \neq \emptyset$ for any i. The colors are used to differentiate between different types of nodes. These subsets are called *color classes* and we write $\pi(v)$ for the color class of a vertex v. For example, in the context of GGP all nodes having the same function share the same color. The order of the color classes is significant, but the order of the vertices within each class is not. We use the coloring to distinguish between vertices with different labels. For example, all vertices in a graph with the label $+$ are in one color class, all vertices with the label $\times$ are in another.

4.2 Isomorphic Graphs and Canonical Labeling Maps

Given a permutation $\gamma : V \to V$ of the vertex set V, let v^γ denote the image of the vertex $v \in V$ under γ. Similar, we define $W^\gamma = \{w^\gamma | w \in W\}$ for $W \subseteq V$ and G^γ as the graph in which vertices v^γ and w^γ are adjacent if and only if v and w are adjacent vertices in G. The image of a coloring π under γ is defined as $\pi^\gamma = (V_1^\gamma, V_2^\gamma, \ldots, V_k^\gamma)$. A permutation γ of a vertex set with coloring π is called *color-preserving* if and only if $\pi^\gamma = \pi$ (i.e., each $v \in V$ has the same color as v^γ). An *automorphism* of a graph G is a color-preserving permutation of the vertices that maps G onto itself. We say that two colored graphs G_1 and G_2 with colorings π_1 and π_2 (having the colors in the same order) are *isomorphic* if and only if there exists a permutation γ such that $G_1^\gamma = G_2$ and $\pi_1^\gamma = \pi_2$. See figure 4 for an example of two isomorphic GP-graphs.

In the following, we make use of *canonical labeling maps*: Given a coloring $\pi = (V_1, V_2, \ldots, V_k)$ of $V = \{0, 1, \ldots, n\}$, we define the canonical coloring $c(\pi)$ as $(\{0, 1, \ldots, |V_1| - 1\}, \{|V_1|, \ldots, |V_1| + |V_2| - 1\}, \ldots, \{n + 1 - |V_k|, \ldots, n\})$. That is, $c(\pi)$ is independent of π except that it has the same number of classes with the same sizes in

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

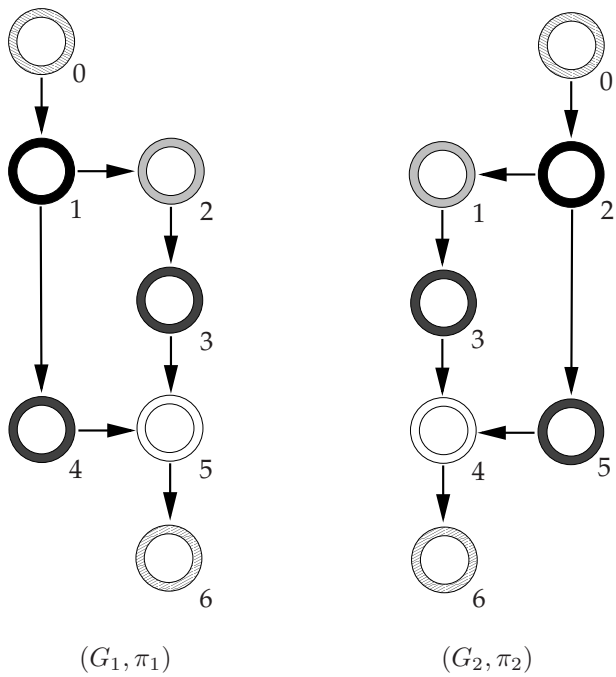

Figure 4: Two isomorphic GP-graphs: (G_1, π_1) is transformed to (G_2, π_2) by switching the vertices 1 and 2 as well as 4 and 5; the corresponding permutation γ is $(12)(45)$ in cyclic notation. The graphs are not automorphic (γ changes the colors of some nodes).

the same order, see figure 5 for an example. A canonical labeling map is a function $\mathcal{C}$ such that for any graph G and coloring π of V

1. there exists some permutation δ such that $\mathcal{C}(G, \pi) = G^\delta$ and $\pi^\delta = c(\pi)$, and

2. for any permutation γ of V we have $\mathcal{C}(G^\gamma, \pi^\gamma) = \mathcal{C}(G, \pi)$.

Such a mapping always exists for the graph spaces under consideration. Roughly speaking, a canonical labeling map $\mathcal{C}$ maps a colored graph G and all graphs isomorphic to G to the same canonical graph that is isomorphic to G. Isomorphism is an equivalence relation and any $\mathcal{C}$ computes a unique representative of the equivalence class of its argument. Formally, the following equivalence holds (McKay, 1981). Suppose the graphs G_1 and G_2 are colored using the same number of vertices of each color. Let π_1 and π_2 denote the corresponding colorings, with the colors in the same order in each. Then $\mathcal{C}(G_1, \pi_1) = \mathcal{C}(G_2, \pi_2)$ if and only if G_1 and G_2 are isomorphic, that is, $G_1^\gamma = G_2$ and $\pi_1^\gamma = \pi_2$ for some permutation γ.

4.3 Counting Isomorphic Graphs

Given a graph G, we want to determine how many different graphs—potential genotypes in GP—exist that are isomorphic to G. Generally, there are $|V|!$ permutations of the vertices. In some applications, vertices belonging to particular color classes must not change their color. That is, we do not allow permutations γ where $\pi(v) \neq \pi(v^\gamma)$ for

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

a vertex v in $\bigcup_{i \in \mathcal{I}} V_i$ for some index set $\mathcal{I} \subseteq \{1, \ldots, k\}$. In this case, there exist

$$p_{\mathcal{I}}(G, \pi) = |\bigcup_{j \notin \mathcal{I}} V_j|! \cdot \prod_{i \in \mathcal{I}} |V_i|!$$

permutations (Igel and Stagge, 2002a).

But not all permutations γ of a graph G generate graphs different from G, some permutations result in exactly the same graph (i.e., $G^\gamma = G$ and $\pi^\gamma = \pi$). Two permutations of a graph G result in the same graph if and only if there exists an automorphism of G that maps one of the permutations onto the other. Let $a(G, \pi)$ denote the number of such automorphisms of G. Then the number of graphs that are isomorphic to G (including G itself) is given by $p_{\mathcal{I}}(G, \pi)/a(G, \pi)$. To see this, suppose two isomorphic graphs G_1 and G_2 with $G_1^{\gamma_1} = G_2$ and $\pi_1^{\gamma_1} = \pi_2$ for a permutation γ_1. Let γ_2 be an automorphism of G_1. Then the concatenation $\gamma_1 \circ \gamma_2$ of the two permutations leads to the same graph as the permutation γ_1 and it holds $G_1^{\gamma_1} = G_1^{\gamma_1 \circ \gamma_2} = G_2$ and $\pi_1^{\gamma_1} = \pi_1^{\gamma_1 \circ \gamma_2} = \pi_2$. There exist $a(G_1, \pi_1) = a(G_2, \pi_2)$ possible choices for γ_2. Thus, for each permutation there exist $a(G_2, \pi_2)$ permutations (including the identity mapping) that lead to the same mapping.

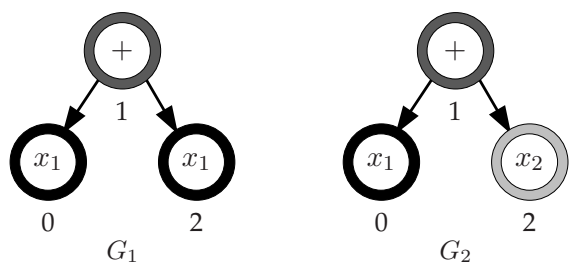

Figure 5: Example graphs $G_1, G_2 = (\{0, 1, 2\}, \{(1, 0), (1, 2)\})$ with different colorings $\pi_1 = (\{0, 2\}, \emptyset, \{1\})$ and $\pi_2 = (\{0\}, \{2\}, \{1\})$, respectively. The graphs are not isomorphic; there is no permutation γ with $\pi_1^\gamma = \pi_2$. Canonical colorings are $c(\pi_1) = (\{0, 1\}, \emptyset, \{2\})$ and $c(\pi_2) = (\{0\}, \{1\}, \{2\})$. For the number of automorphisms, we have $a(G_1, \pi_1) = 2$ and $a(G_2, \pi_2) = 1$.

For example, consider the graph G_2 in figure 5. Then $a(G_2, \pi_2) = 1$, because the identity is the only color-preserving permutation that maps G_2 onto itself, and therefore the number of graphs that are isomorphic to G_2 is $3!/1 = 6$. These six graphs (G, π) are $((\{0, 1, 2\}, \{(1, 0), (1, 2)\}), (\{0\}, \{2\}, \{1\}))$, $((\{0, 1, 2\}, \{(1, 2), (1, 0)\}), (\{2\}, \{0\}, \{1\}))$, $((\{0, 1, 2\}, \{(2, 0), (2, 1)\}), (\{0\}, \{1\}, \{2\}))$, $((\{0, 1, 2\}, \{(0, 2), (0, 1)\}), (\{2\}, \{1\}, \{0\}))$, $((\{0, 1, 2\}, \{(2, 1), (2, 0)\}), (\{1\}, \{0\}, \{2\}))$, and $((\{0, 1, 2\}, \{(0, 1), (0, 2)\}), (\{1\}, \{2\}, \{0\}))$. The graph G_1 has an additional automorphism, namely the permutation that just switches the vertices 0 and 2 (i.e., (02) in cyclic notation), and therefore $a(G_1, \pi_1) = 2$ and the number of isomorphic graphs is $3!/2 = 3$. These graphs are $((\{0, 1, 2\}, \{(1, 0), (1, 2)\}), (\{0, 2\}, \emptyset, \{1\}))$, $((\{0, 1, 2\}, \{(2, 0), (2, 1)\}), (\{0, 1\}, \emptyset, \{2\}))$, and $((\{0, 1, 2\}, \{(0, 1), (0, 2)\}), (\{1, 2\}, \emptyset, \{0\}))$.

4.4 Complexity of Graph Canonization

The general graph isomorphism problem and therefore graph canonization (i.e., canonical labeling) is a potential member of the problem class NPI containing the so called

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

NP-incomplete problems (Garey and Johnson, 1979). The class NPI is defined as NP excluding P and all NP-complete problems. But there exist heuristic algorithms that can handle the canonical labeling for graphs very efficiently. Hence, the time for computing the canonical graph (and the number of automorphisms) can be neglected compared to a fitness evaluation on a real-world problem for graphs of sizes much larger than those in present-day GP systems.

Apart from that, in GP we usually deal with restricted classes of graphs and for certain classes of graphs the isomorphism and canonization problem can be solved efficiently (Köbler, 2006). In particular, the graphs in standard tree-based GP systems are rooted, labeled (colored), ordered trees. For such trees, isomorphisms and canonical representations can be determined in linear time and log-space (Hopcroft and Tarjan, 1974; Lindell, 1992). Buss (1997) could even show that if the trees are represented by strings then the complexity reduces to NC^1. Thus, the overhead of computing the canonical graph can be neglected in tree GP. However, in tree GP the problem of redundant fitness evaluations due to isomorphisms seems to be less prominent than in GGP (as shown by preliminary studies).

In this study, we use the *nauty* software package developed by McKay (1990) for the general graph isomorphism and canonization problem.

5 Canonical Graph Indexed Databases for Speeding Up GP

In this section, we empirically demonstrate the potential of speeding up Genetic Programming by using a fitness database that considers isomorphisms. The GGP system serves as a testbed. Before we present this application, we discuss the relation between graph isomorphisms and neutrality (i.e., non-injective genotype-phenotype mappings) in GP. Albeit not essential to an understanding of the technical aspects of this study, this section serves as a conceptual framework to our study and establishes links to our previous work.

5.1 Isomorphisms and Phenotypic Neutrality

In evolutionary computation, we generally define the phenotype of an individual (more precisely, its *partial* phenotype, which comprises all traits that are of interest, cf. Mahner and Kary, 1997) by the functionality of its genotype. In GP the phenotype of an individual is the behavior (e.g., the input-output or stimulus-action mapping represented by the genotype) of the program it represents. As the phenotype reflects the essential qualities of an individual, the core of an evolutionary algorithm is what phenotypes it generates. This is described by the *search distribution on the phenotype space* $P_{\mathcal{P}}^{(t)}$, which is defined as the probability that a phenotype $p \in \mathcal{P}$ is sampled in generation t. We call $P_{\mathcal{P}}^{(t)}$ the *exploration distribution*. For example, if GP is used for symbolic regression, our main concern is the search behavior in the space of functions, not in the space of symbolic expressions.

Let us assume that there exists a function Γ that maps the genotype of an individual to its phenotype. If the genotype-phenotype mapping Γ is not injective (i.e., multiple genotypes map to the same phenotype), we speak of a *neutral* encoding. The role of neutrality in evolutionary processes has been extensively studied in the context of both natural (e.g., see Kimura, 1968; Conrad, 1990; Schuster, 1996; Huynen, 1996; van Nimwegen, Crutchfield, and Huynen, 1999; Wilke, 2001; Schuster, 2002) and simulated evolution (e.g., see Thierens, 1996; Newman and Engelhardt, 1998; Ebner, 1999; Ebner, Shackleton, and Shipman, 2001; Igel and Stagge, 2002a; Igel and Toussaint, 2003, and references therein). It has been pointed out that the main benefit of neutrality

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

is that it enables the self-adaptation of the search distribution (see Banzhaf, Nordin, Keller, and Francone, 1998; Toussaint and Igel, 2002; Igel and Toussaint, 2003; Toussaint, 2003), which subsumes, for example, the ability to adapt to changing environments (Ebner, Shackleton, and Shipman, 2001). This strategy adaptation is only possible if the genotype-phenotype mapping is *non-trivial*. Informally speaking, a genotype-phenotype mapping (or the neutrality) is non-trivial if the search distribution of an evolutionary algorithm can change just by replacing a genotype in the parent population by a genotype with the same phenotype (see Toussaint, 2003 for a rigorous mathematical treatment of trivial and non-trivial neutrality). In neutral encodings, some phenotypes may be more often represented than others, that is, some phenotypes are frequent, others are rare (e.g., see Schuster, 1996; Igel and Stagge, 2002a). This may introduce a bias that can be counter-balanced based on methods described by Igel and Stagge (2002b).

Graph isomorphisms are usually a source of trivial neutrality. Consider a GP problem where the elements of the function set are all commutative. That is, the order of the children of a vertex does not influence the behavior of the genetic program. For a discussion of this restriction see section 5.3. Then two genotypes that represent graphs that are isomorphic have the same phenotype. The neutrality induced by graph isomorphisms does not usually allow for adaptation of the search distribution. Consider tree GP with standard operators. A mutation operator randomly selects a vertex, the mutation point, and replaces the subtree rooted in the mutation point by a randomly generated tree. A crossover operator randomly selects a vertex in each parent tree, the crossover points, and switches the corresponding subtrees. Assume that the vertices with the same depth in a tree have the same probability to be chosen as crossover or mutation point. In this standard scenario, the exploration distribution does not change if an individual encoding a graph G_1 is replaced in the population by an individual that represents an isomorphic graph G_1^γ. The probability that a vertex v in G_1 is selected as crossover or mutation point is equal to the probability that v^γ in G_1^γ is selected, and the corresponding subtrees are phenotypically the same.

In theory, trivial neutrality can always be eliminated without changing the evolutionary dynamics on the phenotype space (Toussaint, 2003). The trivial neutrality arising from isomorphisms can be removed by transforming the individuals in the initial population and all offspring to their canonical graphs. Thus only canonically labeled graphs occur in the population, that is, each equivalence class of isomorphic individuals is uniquely represented. Without changing the variation operators, this is of little use in practice in standard GP systems (except that one could come up with an efficient compression scheme for the population as the cardinality of the genotype space is reduced).

5.2 Graph Databases

If the fitness evaluation of an individual is very time consuming (e.g., in the applications described by Koza et al., 1999) and the fitness function does not vary over the generations, it is beneficial to store the genotype together with its fitness in a database. If the same genotype is sampled again, its fitness can be looked up. A graph database has successfully been used by Friedrich and Moraga (1996) in the context of evolutionary optimization of neural networks based on a cellular encoding inspired by Gruau (1995). The concept of a genotype database may seem inconsistent with the population paradigm because what is sacrificed is the advantage that evolutionary algorithms keep only a fixed number of solutions in memory. However, in most real-world ap-

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

plications (e.g., in design optimization) the memory requirements and the overhead of using a database can be completely neglected; the limiting factor is solely the time needed for single fitness evaluations.

When the encoding is neutral, it is better to index the database with the phenotype of the individual (given that the phenotype can be described efficiently), because then a single fitness evaluation is sufficient for all genotypes sharing the same phenotype. This leads to the idea to store and look up the canonically labeled graphs in the database when evolving graph structures. Before the quality of a new individual is determined, it can be tested whether the corresponding graph or an isomorphic one was evaluated before by looking up the canonically labeled graph in the database. In that way, more fitness evaluations can be saved. How many depends on the search method used and on the search space. Igel and Stagge (2002b) presented an example in the domain of structure optimization of neural networks, where $21\,\%$ evaluations could *additionally* be saved by storing the fitness of the canonically labeled graph compared to the use of a fitness database indexed with genotypes.

A database indexed with the canonical graph does not consider all kinds of neutrality in GP, only the trivial neutrality due to isomorphisms, see Section 5.1.

5.3 Encoding of GP-graphs

To use the concepts from section 4 (and the software package *nauty* by McKay, 1981) we have to construct an injective function that maps each GP-graph $\tilde{G} = (V_{\text{in}} \cup V_{\text{inner}} \cup V_{\text{out}}, \tilde{E})$ with coloring $\tilde{\pi}$ to a standard graph $G = (V, E)$ with coloring π. For our benchmark problems, a straightforward transformation exists. Basically, we can set $V \leftarrow V_{\text{in}} \cup V_{\text{inner}} \cup V_{\text{out}}$, $\pi \leftarrow \tilde{\pi}$, and $E \leftarrow \{(u, v) \,|\, (u, v) \in \tilde{E}\}$. Still, multiple edges between vertices and the ordering of the edges must be considered and therefore changes to G and π become necessary.

First, we have to handle multiple edges with the same direction between two vertices. Because in all benchmark problems considered here the highest in- or out-degree is two, this can be done by the rule that $e_1, e_2 \in \tilde{E} \wedge e_1 = e_2 = (u, v)$ if and only if $(u, v) \in E$ and (u, v) is the only connection to the $(2, m)$-vertex v and from the $(n, 2)$-vertex u.

Second, we have to be careful with the ordering of edges in $\tilde{G}$. In *GGP* all edges ending at the same vertex are ordered, the same holds for edges leaving a vertex. The order of the input edges is crucial for non-commutative functions and the ordered outward directed edges allow for a deterministic choice of the succeeding vertex (e.g., by pointing to a *then* and an *else* branch in an *if-then-else* statement). Without loss of generality, we only describe our method for keeping the information about the order of edges for an in-degree of two. If there are non-commutative functions, we introduce a new color (i.e., a new vertex type), which we call the order-indicator color.

If a non-commutative function is represented by a $(2, m)$-vertex $v \in V_{\text{inner}}$, we set $V \leftarrow V \cup \{v'\}$, with *pseudo-vertex* v' receiving the order-indicator color. Let $(u, v), (w, v) \in \tilde{E}$ be input edges to v, where u is the first (leftmost) input, then the connections $(u, v), (w, v)$ are replaced by $(u, v'), (v', v), (w, v)$. That is, the leftmost input is tagged by the pseudo-vertex.

If the GP-graphs are mapped to standard graphs as described above, it can be shown that isomorphic GP-graphs have the same phenotype.

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

Task	Method	Isomorphic	Identical
Artificial-Ant	demes, standard	31485.0 (15.74 %)	20939.9 (10.47 %)
Artificial-Ant	no demes, standard	38717.9 (19.36 %)	24173.8 (12.09 %)
Artificial-Ant	no demes, uniform	38343.8 (19.17 %)	23981.7 (11.99 %)
Lawn-Mower	demes, standard	71550.4 (35.78 %)	19260.9 (9.63 %)
Lawn-Mower	no demes, standard	71796.3 (35.90 %)	19478.5 (9.74 %)
Lawn-Mower	no demes, uniform	71180.0 (35.59 %)	19355.4 (9.68 %)
Two-Boxes	demes, standard	41786.8 (20.89 %)	19188.9 (9.59 %)
Two-Boxes	no demes, standard	53018.7 (26.51 %)	22688.9 (11.34 %)
Two-Boxes	no demes, uniform	52901.5 (26.45 %)	22707.6 (11.35 %)

Table 8: Average number (and percentage) of fitness evaluations saved at the end of each trial (i.e., after $2 \cdot 10^5$ evaluations) for the three benchmark problems and the three variants of the GGP algorithm: with demes (demes, standard), without demes using standard initialization (no demes, standard), and without demes starting from a uniformly initialized population (no demes, uniform). Given are the number of evaluations that could be saved using a genotype indexed database (Identical) and a canonical graph indexed database (Isomorphic).

5.4 Empirical Evaluation

The aim of our experiments was to determine how many fitness calculations in GGP are redundant due to isomorphic graphs already examined.

5.4.1 Experiments

We used the data from the experiments above, to assess the performance of GGP. In each trial, we maintained two initially empty databases. In the first database we stored all GP-graphs encountered during evolution. The second database was indexed by the canonically labeled graph corresponding to the GP-graphs encoded as described in section 5. In order to speed up the comparison of graphs, we compressed every graph and saved it in a binary search tree as a triple consisting of the compressed data, the size of the compressed data, and the number of vertices in the graph.

The number of entries in a databases (i.e., the number of different genotypes or the number of different canonical graphs) is equal to the number of fitness evaluations needed if this database would have been used in the EA.

We repeated all experiments without using dynamic demes and different ways of initializing the population. Every trial without demes was done with two different initialization schemes, standard initialization and a uniform initialization with the complete population consisting of copies of a single, randomly chosen genotype.

5.4.2 Results

The results for the three benchmarks problems are summarized in figure 6. In all scenarios, the trivial neutrality induced by isomorphisms is very prominent. There is a large percentage of fitness evaluations of graphs that are isomorphic, but not identical to graphs evaluated before.

A considerable amount of fitness evaluations could be saved by using a graph database. For example, after $2 \cdot 10^5$ evaluations (i.e., after the trials have converged) the fraction of fitness evaluations that could have been saved by using the database indexed by the canonical graphs has increased to 35.8 %, 20.9 %, and 15.7 % for the

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

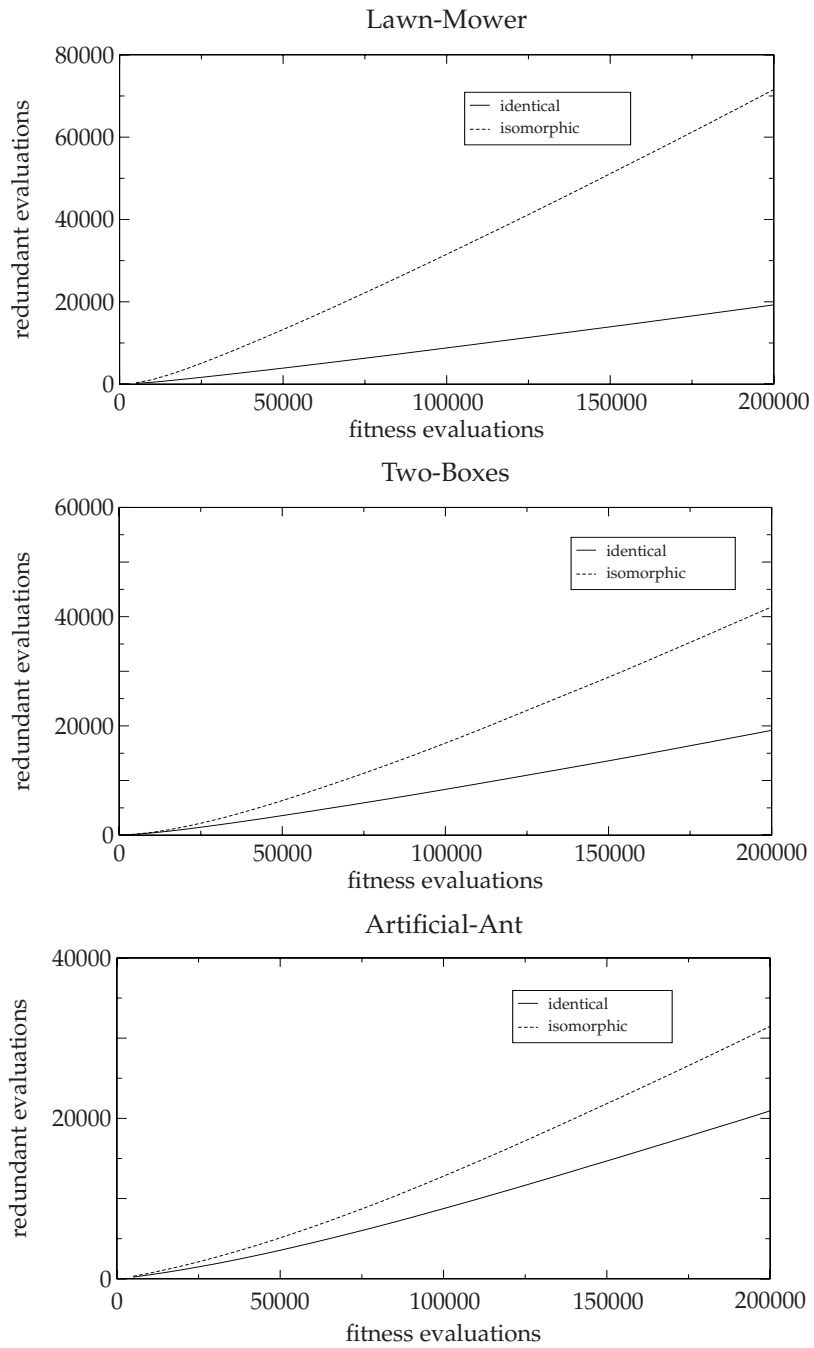

Figure 6: Number of redundant evaluations in the three benchmark problems averaged over 500 trials. The solid lines refer to the number of fitness evaluations that could be saved because the same genotype was evaluated before. The dashed lines additional count the cases where the corresponding canonical graph was evaluated previously.

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

Lawn-Mower, the Two-Boxes, and the Artificial-Ant benchmark, respectively. Of the redundant fitness evaluation, 73.1 %, 54.1 %, and 33.5 %, respectively, were only detected because of using the canonical labeling graph concept. That means, considering the effects of graph isomorphisms has significantly increased the utility of the graph database.

Dynamic demes increase the diversity in the population by introducing new genotypes and they allow access to unexplored areas of the genotype space. In the control experiments without demes the number of fitness evaluations that can be saved using graph databases increases, because more similar graphs are visited. The percentage of evaluations that can be *additionally* saved by exploiting isomorphisms stays approximately the same, that is, the absolute number of evaluations that can additionally be saved using a canonical graph indexed database increases considerably. The different ways of initialization, standard and uniform, lead to almost the same results.

5.5 Factors Influencing the Effects of Isomorphisms

How many additional fitness evaluations are saved by considering graph isomorphisms in a fitness database depends on many aspects. The restrictions on the class of graphs are important. For example, for trees, where the canonization problem can be solved efficiently (see Section 4.4), the effects of isomorphism seem to be less prominent than in the case of GGP. Of course, function and terminal set play a major role. The less different symbols, the higher the probability of observing isomorphic, but not identical graphs. Of course, the more functions out of a function set are commutative, the higher the probability to encounter isomorphic, not identical graphs. But also the operators used and the population size matter. Typical operators, such as adding and deleting nodes, do not produce offspring isomorphic to their parents. If the population size is small and the search is rather local (where "distance" between individuals is measured by the probability that one is generated by the variation operators from the other), considering isomorphism in a fitness database does not save many additional fitness evaluations. A detailed analysis of all these factors is left to further studies.

6 Conclusions

Graph isomorphisms are a source of trivial neutrality in genetic programming (GP). We presented ways of dealing with graph isomorphisms, including how to count the number of isomorphic representations of a graph and how to handle commutative functions. We stressed the use of canonical labeling maps for search algorithms operating on graph spaces.

We believe that fitness databases promise to improve the performance of GP, in particular when applied to real-world applications. The overhead of storing and looking up fitness values in a database can be neglected compared to the time needed for computing the fitness function in many if not most technical and scientific applications of GP. We empirically showed that the number of fitness evaluations can be significantly reduced by means of a fitness database. Depending on the GP system and the problem at hand, considering graph isomorphisms when addressing the database can additionally save evaluations.

The experimental results have been achieved using the GGP algorithm, which has proven to perform very well on common GP benchmark problems. This shows that graph-based GP systems can be beneficial even if the problem at hand does not necessarily require a graph encoding.

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

Acknowledgments

We thank the anonymous reviewers for their helpful suggestions to improve the article. WB acknowledges support from NSERC under Discovery Grant RGPIN 283304-04.

References

Banzhaf, W., P. Nordin, R. E. Keller, and F. D. Francone (1998). *Genetic Programming – An Introduction*. Morgan Kaufmann.

Bruce, W. S. (1997). The lawnmower problem revisited: Stack-based genetic programming and automatically defined functions. In J. R. Koza (Ed.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 52–57. Morgan Kaufmann.

Buss, S. (1997). Alogtime algorithms for tree isomorphism, comparison, and canonization. In *Computational Logic and Proof Theory, 5th Kurt Gödel Colloquium '97*, Volume 1289 of *LNCS*, pp. 18–33. Springer-Verlag.

Chellapilla, K. (1997). Evolutionary programming with tree mutations: Evolving computer programs without crossover. In J. R. Koza (Ed.), *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 431–438. Morgan Kaufmann.

Chickering, D. M. (2002). Learning equivalence classes of Bayesian-network structures. *Journal of Machine Learning Research 2*, 445–498.

Christensen, S. and F. Oppacher (2002). An analysis of Kozas computational effort statistic for genetic programming. In J. A. Foster, E. Lutton, J. Miller, C. Ryan, and A. G. B. Tettamanzi (Eds.), *Genetic Programming: 5th European Conference (EuroGP 2002)*, Volume 2278 of *LNCS*, pp. 182–191. Springer-Verlag.

Conrad, M. (1990). The geometry of evolution. *Biosystems 24*, 61–81.

Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In J. J. Grefenstette (Ed.), *Proceedings of an International Conference on Genetic Algorithms and their Application*, pp. 183–187. Lawrence Erlbaum Associates.

D'haeseleer, P. and J. Bluming (1994). Effects of locality in individual and population evolution. In K. E. Kinnear jr. (Ed.), *Advances in Genetic Programming*. MIT Press.

Ebner, M. (1999). On the search space of genetic programming and its relation to nature's search space. In *IEEE Congress on Evolutionary Computation 1999 (CEC 1999)*, pp. 1357–1361. IEEE Press.

Ebner, M., M. Shackleton, and R. Shipman (2001). How neutral networks influence evolvability. *Complexity 7*(2), 19–33.

Fogel, L. J., A. J. Owens, and M. J. Walsh (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons.

Friedrich, C. M. and C. Moraga (1996). An evolutionary method to find good building-blocks for architectures of artificial neural networks. In *Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU'96)*, Volume 2, pp. 951–956.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability, A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Company.

Gruau, F. (1995). Automatic definition of modular neural networks. *Adaptive Behavior 3*(2), 151–183.

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

Heywood, M. I. and A. N. Zincir-Heywood (2002). Dynamic page based crossover in linear genetic programming. *IEEE Transactions on Systems, Man, and Cybernetics: Part B 32*(3), 380–388.

Hopcroft, J. and R. Tarjan (1974). Efficient planarity testing. *Journal of the ACM 21*(4), 549–568.

Huynen, M. A. (1996). Exploring phenotype space through neutral evolution. *Journal of Molecular Evolution 43*, 165–169.

Igel, C. and P. Stagge (2002a). Effects of phenotypic redundancy in structure optimization. *IEEE Transactions on Evolutionary Computation 6*(1), 74–85.

Igel, C. and P. Stagge (2002b). Graph isomorphisms affect structure optimization of neural networks. In *International Joint Conference on Neural Networks 2002 (IJCNN)*, pp. 142–147. IEEE Press.

Igel, C. and M. Toussaint (2003). Neutrality and self-adaptation. *Natural Computing 2*(2), 117–13.

Kantschik, W. and W. Banzhaf (2002). Lineargraph GP - a new GP structure. In E. Lutton, J. Foster, J. Miller, C. Ryan, and A. Tettamanzi (Eds.), *Genetic Programming, Proceedings of 5th EuroGP*, Volume 2278 of *LNCS*, pp. 83–92. Springer.

Kimura, M. (1968). Evolutionary rate at the molecular level. *Nature 217*, 624–626.

Köbler, J. (2006). On graph isomorphism for restricted graph classes. In A. Beckmann, U. Berger, B. Löwe, and J. V. Tucker (Eds.), *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006*, Volume 3988 of *LNCS*, pp. 241–256. Springer-Verlag.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press.

Koza, J. R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

Koza, J. R., F. H. Bennett III, D. Andre, and M. A. Keane (1999). *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers.

Kuscu, I. (1998). Artificial ant problem revisited. In V. W. Porto (Ed.), *Seventh Annual Conference on Evolutionary Programming*, Volume 1447 of *LNCS*, pp. 799–808. Springer-Verlag.

Langdon, W. B. (1998). Better trained ants. In R. Poli (Ed.), *Late Breaking Papers at EuroGP'98: The First European Workshop on Genetic Programming*, pp. 11–13. The University of Birmingham.

Langdon, W. B. and R. Poli (1998). Why ants are hard. In J. R. Koza (Ed.), *Genetic Programming 1998: Procceedings of the Third Annual Conference*, pp. 193–201. Morgan Kaufmann.

Lindell, S. (1992). A logspace algorithm for tree canonization. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing (STOC)*, pp. 400–404. ACM Press.

Luke, S. and L. Panait (2002). Is the perfect the enemy of the good? In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002*, pp. 820–828. Morgan Kaufmann.

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

Mahner, M. and M. Kary (1997). What exactly are genomes, genotypes and phenotypes? And what about phenomes? *Journal of Theoretical Biology 186*(1), 55–63.

McKay, B. D. (1981). Practical graph isomorphism. *Congressus Numerantium 30*, 47–87.

McKay, B. D. (1990). *nauty* user's guide (version 1.5). Technical Report TR-CS-90-02, Australian National University, Computer Science Department, Canberra, Australia.

Newman, M. E. J. and R. Engelhardt (1998). Effects of neutral selection on the evolution of molecular species. *Proceedings of the Royal Society of London, Series B: Biological Sciences 265*(1403), 1333–1338.

Niehaus, J. (2003). Graphbasierte Genetische Programmierung. Dissertation, Chair of System Analysis, Computer Science Department, Dortmund University, Dortmund, Germany. `http://de.scientificcommons.org/866201`.

Niehaus, J. and W. Banzhaf (2001). Adaption of operator probabilities in genetic programming. In J. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. G. B. Tettamanzi, and W. B. Langdon (Eds.), *Genetic Programming: 4th European Conference (EuroGP 2001)*, Volume 2038 of *LNCS*, pp. 325–336. Spring-Verlag.

Niehaus, J. and W. Banzhaf (2003). More on computational effort statistic for genetic programming. In C. Ryan, T. Soule, M. Keijzer, E. P. K. Tsang, R. Poli, and E. Costa (Eds.), *Genetic Programming: 6th European Conference (EuroGP 2003)*, Volume 2610 of *LNCS*, pp. 164–172. Springer-Verlag.

O'Neill, M., C. Ryan, M. Keijzer, and M. Cattolico (2003). Crossover in grammatical evolution. *Genetic Programming and Evolvable Machines 4*(1), 67–93.

Poli, R. (1997). Evolution of graph-like programs with parallel distributed genetic programming. In T. Bäck (Ed.), *Genetic Algorithms: Proceedings of the Seventh International Conference*, pp. 345–353. Morgan Kaufmann.

Schuster, P. (1996). Landscapes and molecular evolution. *Physica D 107*, 331–363.

Schuster, P. (2002). Molecular insights into evolution of phenotypes. In J. P. Crutchfield and P. Schuster (Eds.), *Evolutionary Dynamics – Exploring the Interplay of Accident, Selection, Neutrality and Function*, Santa Fe Institute Series in the Science of Complexity. Oxford University Press.

Spector, L. and S. Luke (1996). Cultural transmission of information in gp. In J. R. K. et.al (Ed.), *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 209–214. MIT Press.

Stagge, P. and C. Igel (2000). Neural network structures and isomorphisms: Random walk characteristics of the search space. In X. Yao and D. B. Fogel (Eds.), *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (ECNN 2000)*, pp. 82–90. IEEE Press.

Stagge, P. and C. Igel (2001). Structure optimization and isomorphisms. In L. Kallel, B. Naudts, and A. Rogers (Eds.), *Theoretical Aspects of Evolutionary Computing*, Natural Computing series, pp. 409–422. Springer-Verlag.

Syswerda, G. (1991). A study of reproduction in generational and steady state genetic algorithms. In G. J. E. Rawlins (Ed.), *Foundations of Genetic Algorithms (FOGA I)*, pp. 94–101. Morgan Kaufmann Publishers.

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021

Teller, A. and M. Veloso (1995). Program evolution for data mining. *The International Journal for Expert Systems 8*(3), 216–236.

Terrio, M. D. and M. D. Heywood (2002). Directing crossover for reduction of bloat in gp. In *Proceedings of the 2002 IEEE Canadian Conference on Electrical & Computer Engineering*, pp. 1111–1115. IEEE Press.

Thierens, D. (1996). Non-redundant genetic coding of neural networks. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pp. 571–575. IEEE Press.

Toussaint, M. (2003). On the evolution of phenotypic exploration distributions. In C. Cotta, K. De Jong, R. Poli, and J. Rowe (Eds.), *Foundations of Genetic Algorithms 7 (FOGA VII)*, pp. 169–182. Morgan Kaufmann.

Toussaint, M. and C. Igel (2002). Neutrality: A necessity for self-adaptation. In *IEEE Congress on Evolutionary Computation 2002 (CEC 2002)*, pp. 1354–1359. IEEE Press.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind 59*, 433–46.

van Nimwegen, E., J. P. Crutchfield, and M. Huynen (1999). Neutral evolution of mutational robustness. *Proceedings of the National Academy of Sciences 96*, 9716–9720.

Whitley, L. D. (1989). The GENITOR algorithm and selection pressure: Why rank–based allocation of reproductive trials is best. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89), Jul 4–7*, pp. 116–121. George Mason University: Morgan Kaufmann.

Wilke, C. O. (2001). Adaptive evolution on neutral networks. *Bulletin of Mathematical Biology 63*(4), 715–730.

Downloaded from http://www.mitpressjournals.org/doi/pdfplus/10.1162/evco.2007.15.2.199 by guest on 18 April 2021