Evolutionary Computation Just Accepted MS.
https://doi.org/10.1162/evco_a_00025
© 2025 by the Massachusetts Institute of Technology

Cross-Representation Genetic Programming: A Case Study on Tree-based and Linear Representations

Zhixing Huang zhixing.huang@ecs.vuw.ac.nz
Yi Mei yi.mei@ecs.vuw.ac.nz
Fangfang Zhang✉ fangfang.zhang@ecs.vuw.ac.nz
Mengjie Zhang mengjie.zhang@ecs.vuw.ac.nz
Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, Wellington, 6140, New Zealand

Wolfgang Banzhaf banzhafw@msu.edu
Department of Computer Science and Engineering, BEACON Center for the Study of Evolution in Action, and Ecology, Evolution and Behavior Program, Michigan State University, East Lansing, MI 48864, USA

Abstract

Existing genetic programming (GP) methods are typically designed based on a certain representation, such as tree-based or linear representations. These representations show various pros and cons in different domains. However, due to the complicated relationships among representation and fitness landscapes of GP, it is hard to intuitively determine which GP representation is the most suitable for solving a certain problem. Evolving programs (or models) with multiple representations simultaneously can alternatively search on different fitness landscapes since representations are highly related to the search space that essentially defines the fitness landscape. Fully using the latent synergies among different GP individual representations might be helpful for GP to search for better solutions. However, existing GP literature rarely investigates the simultaneous effective evolution of multiple representations. To fill this gap, this paper proposes a cross-representation GP algorithm based on tree-based and linear representations, which are two commonly used GP representations. In addition, we develop a new cross-representation crossover operator to harness the interplay between tree-based and linear representations. Empirical results show that navigating the learned knowledge between basic tree-based and linear representations successfully improves the effectiveness of GP with solely tree-based or linear representation in solving symbolic regression and dynamic job shop scheduling problems.

Keywords

Cross-representation, Tree-based genetic programming, Linear genetic programming, Symbolic regression, Dynamic job shop scheduling.

1 Introduction

Genetic programming (GP) has shown impressive performance in many machine learning domains such as classification [4, 27] and symbolic regression [22, 43]. Over the years, many GP variants have been proposed, such as tree-based GP (TGP)[26], Linear

GP (LGP) [32, 33], Cartesian GP [29], gene expression programming [14], graph-based genetic programming [2], and grammar-guided GP [15]. Traditionally, GP only evolves individuals within a single representation. The individual representation (and its corresponding search mechanism) directly defines the search space and the corresponding fitness landscape.

Generally speaking, a GP representation is expected to be suitable for only a subset of problems. Although some existing studies have investigated the performance of different GP representations in solving different problems based on empirical comparisons [40, 42], extending such kind of knowledge to unseen domains is difficult, and such investigations are often too time-consuming and it is hard to cover all different branches and variants of a problem. When encountering an emerging application or a new problem, users have scarce domain knowledge in selecting a GP representation. To address the risk in selecting inappropriate representations, simultaneously evolving multiple representations is a possible way.

However, simultaneously evolving multiple representations is non-trivial. Evolving multiple representations independently renders each representation unable to make full use of all the computation resources to search for effective solutions. Our empirical results also show that evolving multiple representations independently only performs similarly to baselines (see Section 4.4.1). On the other hand, different GP representations cannot exchange building blocks directly since existing genetic operators for a certain representation cannot accept parents with another representation.

This paper proposes a new Cross-Representation GP (CRGP) algorithm that simultaneously evolves individuals with more than one representation. This paper focuses on the CRGP with two typical GP representations, tree-based (i.e., TGP) [26] and linear-based (i.e., LGP) [5] representations, denoted as CRGP-TL. TGP and LGP have very different program representations, TGP with tree-based structures, and LGP with instruction lists. Consequently, the structures of TGP programs are usually wide, while the topological structures of LGP programs are usually long and narrow [20]. It is likely that the two GP representations and their corresponding search mechanisms lead to different fitness landscapes. CRGP-TL simultaneously evolves sub-populations with tree-based and linear GP representations for a single task and exchanges search information across representations, aiming to obtain more diverse useful building blocks with a higher chance to find better solutions. CRGP-TL includes a novel adjacency list-based crossover to effectively exchange building blocks between tree-based and linear GP representations. An adjacency list is a common and universal representation of graphs and can represent different kinds of topological structures. Besides, adjacency lists can convey graph information such as the frequency of different nodes and their connections.

Although adjacency lists can be an intermediate representation for tree-based and linear representations, this does not mean that an adjacency list is a more effective GP representation than tree-based or linear representations because of the two following reasons. First, existing literature shows that evolving computer programs based on graph-based structures is not always better than tree-based and linear representations [40]. Different representations have their own pros and cons for different tasks. Second, a conventional adjacency list relies on graph node indices to distinguish graph nodes. But different graphs (i.e., GP individuals) likely have different indices for the same building blocks (e.g., a three-node building block "$x_1 + x_2$" might be indexed as "$\mathcal{A} \rightarrow [\mathcal{B}, \mathcal{C}]$" and "$\mathcal{D} \rightarrow [\mathcal{E}, \mathcal{F}]$" in two different GP individuals). Thus, it is difficult for conventional adjacency lists from different graphs to represent the same building

Evolutionary Computation Just Accepted MS.
https://doi.org/10.1162/evco_a_00025

blocks when the graphs use different indices.

This paper has three main contributions:

1. We propose a cross-representation evolutionary framework for GP methods. The proposed evolutionary framework simultaneously evolves multiple sub-populations, each with a distinct GP representation and all solving the same task. By simultaneously evolving multiple representation, the cross-representation evolutionary framework reduces the risk of choosing inappropriate representations and stimulates GP to search for more effective solutions To the best of our knowledge, this paper is the first to highlight that cooperation among different GP representations is beneficial for GP search performance.

2. To implement the new evolutionary framework, we propose a CRGP algorithm based on two representative GP systems (i.e., TGP and LGP), denoted as CRGP-TL. The newly proposed algorithm simultaneously evolves two sub-populations, one with tree-based representation and the other with linear representation. A new crossover operator is introduced based on the adjacency list to exchange building blocks from tree-based and linear subpopulations in the course of evolution.

3. This paper verifies the effectiveness of CRGP-TL by two substantially different applications. The two applications are symbolic regression and automatic design of decision rules in dynamic job shop scheduling problems, which cover a wide range of applications [6, 10, 38, 39]. The results show that simultaneously evolving basic tree-based and linear representations is more effective than the original single-representation methods in both problem domains. Furthermore, by extending the cross-representation evolutionary framework to other advanced methods, we got significant performance improvement for dynamic job shop scheduling problems.

2 Literature Review

2.1 Tree-based and Linear GP Representations

TGP [26] uses tree-based representation, where each individual encodes a computer program as a tree. Every tree node represents a function or a terminal (i.e., input feature). Function nodes accept inputs from their sub-trees and deliver results to their parent nodes. Each tree node has up to one parent node. All intermediate results from sub-trees are aggregated at the root, with the root outputting the final result of the program. Tree-based representation has been successfully applied to different domains [3, 36, 48].

LGP represents a computer program by a sequence of register-based instructions [5]. In LGP, every instruction f in the instruction sequence $\mathbf{F} = \{f_1, f_2, ...f_{|F|}\}$ manipulates registers from the same set of registers $\mathbb{R} = \{\mathrm{R}[0], \mathrm{R}[1], \cdots, \mathrm{R}[|\mathbb{R}|-1]\}$, based on the operation in the instruction (denoted as f_{fun}). The registers in f can be categorized into destination registers (f_d) and source registers (f_s). In our work, there are at most one destination register and two source registers (denoted as $f_{s,1}$ and $f_{s,2}$) in each instruction. The final outputs of LGP programs are stored in designated destination registers, normally starting from the first register, R[0], by default. An LGP program can be decoded into a graph. By connecting the operations in the instructions based on registers, the instruction sequence can be decoded into a directed acyclic graph (DAG), in which every graph node can have more than one parent. A directed edge points from a certain graph node to another providing inputs. A comparison between a linear representation and a tree-based representation for the same mathematical formula

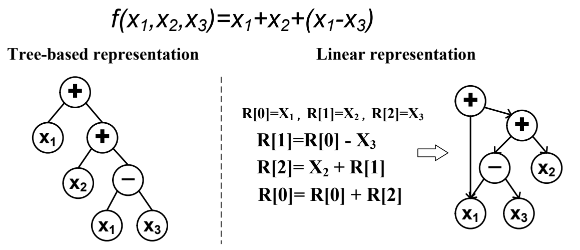
f(x1,x2,x3)=x1+x2+(x1-x3)
Tree-based representation
Linear representation
R[0]=X1 , R[1]=X2 , R[2]=X3
R[1]=R[0] - X3
R[2]= X2 + R[1]
R[0]= R[0] + R[2]

Figure 1: An example of GP individuals with tree-based and linear representations for the same mathematical formula.

"$f(x_1, x_2, x_3) = x_1 + x_2 + (x_1 - x_3)$" is shown in Fig. 1. Specifically, in the linear representation, x_1 to x_3 are read-only input registers, and the calculation registers $R[0]$ to $R[2]$ are initialized by x_1 to x_3 respectively (e.g., the first instruction is equivalent to "$R[1] = x_1 - x_3$"). The final output of the instruction sequence is stored in $R[0]$. The DAG of the LGP individual is also shown in Fig. 1.

Existing literature has shown that different GP representations have superior performance in different domains [25, 40, 42]. For example, tree-based representations are good at parallelizing the computation of different building blocks (i.e., different sub-trees), which benefits their performance on even parity problems and synthetic symbolic regression problems with many distinct sub-programs. On the contrary, the linear representation is good at reusing building blocks by registers, which enhances its performance in real-world symbolic regression problems and dispatching rules design problems where many sub-programs can be reused [18, 40]. However, to the best of our knowledge, all of the GP methods evolve GP individuals only with one unified representation which essentially defines the search space and the corresponding fitness landscape. There is no existing literature that uses the search spaces from different representations to enhance the GP search for an optimization problem. Since it is not guaranteed that the chosen representation is effective for a specific problem, evolving multiple potential representations simultaneously reduces the risks of inadequate GP representations and is expected to improve GP performance.

2.2 Enhancing Evolution By Switching Fitness Landscapes

A fitness landscape consists of three components: search space, fitness function, and neighborhood function [37]. Particularly, the search space of GP is the set of all possible solutions defined by a GP representation, the fitness function evaluates how good the GP individuals are, and the neighborhood functions are the genetic operators producing offspring based on parents. This paper simultaneously evolves the GP individuals with multiple different representations, which is similar to simultaneously searching on multiple related fitness landscapes (as GP individual representations are highly related to fitness landscapes). In the existing literature, there have been studies [41, 44] showing that making full use of related fitness landscapes in designing search mechanisms is an effective way to improve search performance.

Multitask optimization [35] is an example of an optimization method that enhances search performance by mutually exchanging information among the fitness landscapes with similar fitness functions. Specifically, multitask optimization constructs similar fitness functions by simultaneously solving several similar tasks. For example, Gupta et al. [16] used a multitask evolutionary computation method (i.e., multifactorial evolutionary algorithm) to simultaneously solve multiple continuous and discrete optimization problems whose optima are close in the search space. Yi et al. [45] further confirmed that searching for solutions to similar tasks is also effective in solving combinatorial optimization problems.

Sharing the GP search information among tasks is helpful to GP evolution. For example, Zhong et al. [50] proposed a multifactorial gene expression programming method to solve more than one symbolic regression problem simultaneously. Zhang et al. [46] compared the effectiveness of different evolutionary multitask frameworks in solving dynamic flexible job shop scheduling and extended the multitask frameworks to multiobjective optimization in flexible dynamic job shop scheduling [47]. Huang et al. [23] investigated the effectiveness of LGP in existing evolutionary multitask frameworks for solving dynamic job shop scheduling.

The so-called multiform optimization is another way to build up similar fitness landscapes. The multiform optimization simultaneously optimizes several alternative formulations for a single problem. For example, Da et al. [9] formulated a traveling salesman problem into a single-objective and a multiobjective optimization problem respectively, and solved the two optimization problems via a multitask optimization method. Since multiobjective formulations often introduce plateaus into fitness landscapes by evaluating solutions from multiple perspectives, the multiobjective optimization task is expected to remove some local optima from the single-objective formulation. Additional formulations can also be constructed by adding or relaxing constraints on the optimization problem [24], which is equivalent to constructing different search spaces (and hence different fitness landscapes) for solving the single task.

Changing neighborhood functions to reshape the fitness landscape can construct related fitness landscapes in the search for effective solutions. One representative example is variable neighborhood search [30], which switches neighborhood functions in the course of search. By searching within different neighborhoods, variable neighborhood search can reach distant solutions via local search and has a better chance to jump out from a local optimum. Variable neighborhood search is an effective strategy to enhance other search techniques [7].

All of these existing studies show that designing search mechanisms based on fitness landscape considerations is beneficial to search performance. However, using different solution representations to construct related fitness landscapes is not well investigated when solving the same task, especially in the GP area. Although some studies of evolutionary multitask optimization use different solution representations for different tasks [12, 13], their solution representations are mainly designed based on problem-specific decision variables, which is much more intuitive than designing GP representations. Further, the solutions in most evolutionary computation studies are numerical and have a simple neighborhood function. This is vastly different from GP representations that are symbolic and might have neighborhoods exponentially increasing with program size.

3 Cross-representation Genetic Programming with TGP and LGP

3.1 Overall Framework

We propose an overall framework of the CRGP, as shown in Fig. 2 (with the new components highlighted in grey). In contrast to the evolutionary framework of basic GP methods, CRGP evolves multiple sub-populations, each evolving a unique GP representation. When breeding offspring, CRGP selects parents from all the representations and also applies cross-representation genetic operators to produce offspring. Offspring

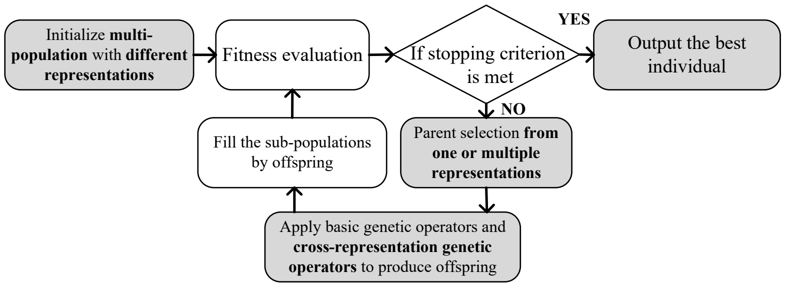
Initialize multi-population with different representations
Fitness evaluation
If stopping criterion is met
YES
Output the best individual
NO
Fill the sub-populations by offspring
Parent selection from one or multiple representations
Apply basic genetic operators and cross-representation genetic operators to produce offspring

Figure 2: Evolutionary framework of CRGP. The novel components are highlighted by the dark boxes.

Algorithm 1: CRGP-TL

```
Input: cross-representation crossover rate $\theta_t$, tournament selection size $s$, maximum depth of the
       tree $\overline{d}$, maximum number of instructions $\overline{L}$, minimum number of instruction $\underline{L}$
Output: best individual $\mathbf{h}$
1  Initialize two sub-populations, $\mathbb{S}_1$ for the tree-based representation and $\mathbb{S}_2$ for the linear
     representation.
2  while stopping criteria are not satisfied do
       // Evaluation
3      Evaluate fitness of individuals $\forall \mathbf{f} \in \mathbb{S}_1 \bigcup \mathbb{S}_2$.
4      Update the best individuals $\mathbf{h}$ in $\mathbb{S}_1 \bigcup \mathbb{S}_2$;
5      for $j \leftarrow 1$ to $2$ do
6          $\mathbb{S}'_j \leftarrow \emptyset$;
7          Clone top-1% individuals of $\mathbb{S}_j$ into $\mathbb{S}'_j$;
8          while $|\mathbb{S}'_j| < |\mathbb{S}_j|$ do
9              $rnd \leftarrow \texttt{rand}(0,1)$;
10             if $rnd < \theta_t$ then
11                 $\mathbf{p}_1 \leftarrow \texttt{TournamentSelection}(\mathbb{S}_j, s)$;
12                 $i \leftarrow \texttt{randint}(1,2)$;
13                 $\mathbf{p}_2 \leftarrow \texttt{TournamentSelection}(\mathbb{S}_i, s)$;
14                 $\mathbf{c} \leftarrow \texttt{CALX}(\mathbf{p}_1, \mathbf{p}_2, \overline{d}, \overline{L}, \underline{L})$;
15             else
16                 Apply corresponding (i.e., TGP or LGP) basic genetic operators on $\mathbb{S}_j$ to produce
                     offspring $\mathbf{c}$ (or $\mathbf{c}_1$ and $\mathbf{c}_2$);
17             $\mathbb{S}'_j \leftarrow \mathbb{S}'_j \bigcup \{\mathbf{c}\}$ (or $\mathbb{S}'_j \leftarrow \mathbb{S}'_j \bigcup \{\mathbf{c}_1, \mathbf{c}_2\}$);
18         $\mathbb{S}_j \leftarrow \mathbb{S}'_j$;
19 Return $\mathbf{h}$.
```

of a certain representation fill the corresponding sub-population of the next generation. After a predefined number of generations, the best solution is output. Note that the best solution is either from the tree-based representation or the linear representation.

This paper studies CRGP based on the TGP and LGP, denoted as CRGP-TL. The pseudo-code of CRGP-TL is shown in Alg. 1 [1]. First, CRGP-TL initializes two sub-populations, one for evolving tree-based GP individuals and the other for evolving LGP individuals. All individuals in these two sub-populations evolve simultaneously. For each sub-population, we perform elitism selection to retain elite individuals for

[1] `rand`(a, b) returns a random floating-point number in $[a, b)$. `randint`(a, b) returns a random integer number in $[a, b]$. $|\cdot|$ denotes the cardinality of a container (e.g., set or list). $(\cdot)$ following a container denotes getting an element from the container based on the index.

Evolutionary Computation Just Accepted MS.
https://doi.org/10.1162/evco_a_00025

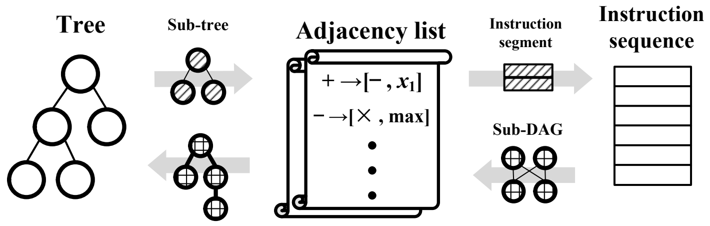

Figure 3: The schematic diagram of CALX between trees and instruction sequences

the next generation (line 7). To fill the sub-population of the next generation, we use tournament selection (i.e., `TournamentSelection`$(\cdot)$) to select individuals as parents and apply different genetic operators based on predefined rates. Specifically, CRGP-TL triggers the cross-representation adjacency-list based crossover (`CALX`$(\cdot)$) based on a predefined rate θ_t (line 10). If the cross-representation adjacency-list-based crossover is triggered, CRGP-TL selects a parent from the current sub-population and selects the other parent from one of the two sub-populations. `CALX`$(\cdot)$ accepts the two parents and produces an offspring. If the operator is not triggered, CRGP-TL applies basic TGP or LGP genetic operators to evolve tree-based and linear representations separately (lines 15-16). The newly generated offspring form the new populations with different representations (line 17). The evolution continues until a stopping criterion is met. The best individual among all the sub-populations with different representations is output as the final result.

3.2 Cross-representation Adjacency List-based Crossover

Knowledge transfer among representations is implemented by the cross-representation adjacency list-based crossover (CALX), as shown in Fig. 3. To swap genetic materials, a tree or an instruction sequence first selects a sub-tree or an instruction segment. The instruction segment is essentially a sub-DAG (or multiple disconnected sub-DAGs). The sub-tree and sub-DAGs are further converted into adjacency lists[2]. Based on the representation of the recipient, a new sub-tree or instruction segment is constructed based on the adjacency list and swapped into the recipient.

An adjacency list is a high-level representation of a graph. This paper denotes an adjacency list as

$$\mathbf{L} = \big(\ [fun_1, \mathbf{A}_1] \quad [fun_2, \mathbf{A}_2] \quad \cdots \quad [fun_{|\mathbf{L}|}, \mathbf{A}_{|\mathbf{L}|}]\ \big)$$

where each item $[fun_i, \mathbf{A}_i]$ specifies a function fun_i and its adjacent nodes $\mathbf{A}_i$. Specifically, $\mathbf{A}_i$ contains one or two nodes in this paper since we only consider unary and binary functions. For example, we convert the left tree in Fig. 1 as

$$\mathbf{L} = \big(\ [+, [x_1, +]] \quad [+, [x_2, -]] \quad [-, [x_1, x_3]]\ \big)$$

It is worth noting that the adjacency list in this paper uses primitive symbols (i.e., functions or terminals) to specify graph nodes to highlight building blocks, which is different from conventional adjacency lists which distinguish graph nodes by the indexes.

[2]Disconnected graph nodes are converted into adjacency lists with empty adjacent nodes $\mathbf{A}$

Algorithm 2: `CALX`

```
Input: Parent individuals $\mathbf{p}_1$ and $\mathbf{p}_2$, maximum depth of the tree $\overline{d}$, maximum number of
       instructions $\overline{L}$, minimum number of instruction $\underline{L}$
Output: An offspring $\mathbf{c}$
1  Clone $\mathbf{p}_1$ as $\mathbf{c}$;
2  if $\mathbf{p}_1$ is a TGP individual then
       // breeding trees based on adjacency lists
3      Randomly pick an inner tree node $t_1$ from $\mathbf{c}$;
4      if $\mathbf{p}_2$ is a TGP individual then
5          Randomly pick an inner tree node $t_2$ from $\mathbf{p}_2$;
6          $\mathbf{L} \leftarrow$ get the adjacency list of the sub-tree in $\mathbf{p}_2$ whose root is $t_2$;
7      else if $\mathbf{p}_2$ is an LGP individual then
8          Randomly select a crossover point $t_2$ and select an instruction segment
             $\mathbf{F}' \subseteq [\mathbf{p}_2(t_2), \mathbf{p}_2(|\mathbf{p}_2|)]$;
9          $\mathbf{L} \leftarrow$ get the adjacency list of the sub-graph from $\mathbf{F}'$;
10     $t_1' \leftarrow$ GrowTreeBasedAL($\mathbf{L}$, the depth of $t_1$ in $\mathbf{c}$, $1, \overline{d}$);
11     Replace the sub-tree with the root of $t_1$ as the sub-tree with the root of $t_1'$ in $\mathbf{c}$;
12 else if $\mathbf{p}_1$ is an LGP individual then
       // breeding instructions based on adjacency lists
13     if $\mathbf{p}_2$ is a TGP individual then
14         Randomly select a crossover point $t_1$ and select an instruction segment
             $\mathbf{F}_1' \subseteq [\mathbf{c}(1), \mathbf{c}(t_1)]$;
15         $\mathbf{L}_1 \leftarrow$ get the adjacency list of the sub-graph from $\mathbf{F}_1'$;
16         Randomly pick an inner tree node $t_2$ from $\mathbf{p}_2$;
17         $\mathbf{L}_2 \leftarrow$ get the adjacency list of the sub-tree in $\mathbf{p}_2$ whose root is $t_2$;
18     else if $\mathbf{p}_2$ is an LGP individual then
19         Randomly select a crossover point $t_1$ and select an instruction segment
             $\mathbf{F}_1' \subseteq [\mathbf{c}(t_1), \mathbf{c}(|\mathbf{c}|)]$;
20         $\mathbf{L}_1 \leftarrow$ get the adjacency list of the sub-graph from $\mathbf{F}_1'$;
21         Randomly select a crossover point $t_2$ and select an instruction segment
             $\mathbf{F}_2' \subseteq [\mathbf{p}_2(t_2), \mathbf{p}_2(|\mathbf{p}_2|)]$;
22         $\mathbf{L}_2 \leftarrow$ get the adjacency list of the sub-graph from $\mathbf{F}_2'$;
23     $\mathbf{c} \leftarrow$ GrowInstructionBasedAL($\mathbf{p}_1, \mathbf{L}_1, \mathbf{L}_2, t_1, n_1$)
24     if $|\mathbf{c}| \notin [\overline{L}, \underline{L}]$ then
25         $\mathbf{c} \leftarrow \mathbf{p}_1$;
26 Return $\mathbf{c}$;
```

3.2.1 Breeding Trees Based on Adjacency Lists

The pseudo-code of `CALX`$(\cdot)$ is shown in Alg.2. If the first and second parents are both TGP individuals, an inner tree node t_2 is randomly selected from the second parent, and an adjacency list $\mathbf{L}$ is generated based on the sub-tree under t_2 (lines 3-6). If the first parent is a TGP individual and the second parent is an LGP individual, we randomly select an instruction segment $\mathbf{F}'$ (lines 7-8) and convert it to sub-DAGs. $\mathbf{L}$ is further constructed based on the selected sub-graphs (line 9). Then, we apply `GrowTreeBasedAL`$(\cdot)$ to build a sub-tree based on $\mathbf{L}$, as shown in Alg. 3.

`GrowTreeBasedAL`$(\cdot)$ is a recursive function to construct tree nodes based on $\mathbf{L}$. Specifically, if `GrowTreeBasedAL`$(\cdot)$ accepts an empty $\mathbf{L}$ or has reached the maximum depth, it returns a random sub-tree to ensure the validity (lines 1-2). Otherwise, `GrowTreeBasedAL`$(\cdot)$ grows a tree node r based on $\mathbf{L}$ (line 3). If r is a function, `GrowTreeBasedAL`$(\cdot)$ checks the adjacency list and recursively applies `GrowTreeBasedAL`$(\cdot)$ to grow the sub-trees of r (lines 4-13). Random sub-trees are constructed if there are no consistent entities in $\mathbf{L}$ (lines 11-12).

Fig. 4 is an example of constructing a tree based on an adjacency list. The first

Algorithm 3: `GrowTreeBasedAL`

```
Input: Adjacency list $\mathbf{L}$, current depth $d$, index of $\mathbf{L}$ $I$, maximum depth of the tree $\overline{d}$
Output: A tree root $r$
1  if $|\mathbf{L}| = 0$ or $d = \overline{d}$ then
2      Return $r \leftarrow$ a random sub-tree whose depth $\leq \overline{d} - d + 1$;
3  $[r, \mathbf{A}] \leftarrow \mathbf{L}(I)$;
4  if $r$ is a function then
5      for $j \leftarrow 1$ to $|\mathbf{A}|$ do
6          $c' \leftarrow \mathbf{A}(j)$;
7          if $c'$ is a function then
8              $\mathbf{L}' \leftarrow$ collect the entities from $\mathbf{L}(k), k \in [I, |\mathbf{L}|]$ with $\mathbf{L}(k).fun = c'$;
9              if $\mathbf{L}' \neq \emptyset$ then
10                 $c' \leftarrow$ GrowTreeBasedAL($\mathbf{L}, d+1,$ randint$(1, |\mathbf{L}'|), \overline{d}$);
11             else
12                 $c' \leftarrow$ a random sub-tree whose depth $\leq \overline{d} - d - 1$;
13         Append $c'$ as $r$'s child;
14 Return $r$;
```

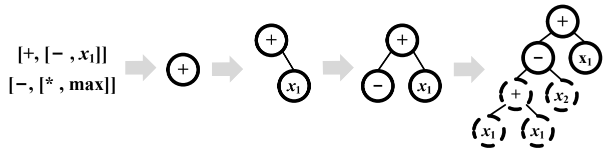

Figure 4: An example of constructing a tree by `GrowTreeBasedAL`$(\cdot)$. The dashed tree nodes are randomly generated.

item in the adjacency list is "$[+, [-, x_1]]$", and hence the root node of the new sub-tree is "$+$". Since the adjacent nodes of "$+$" are "$-$" (a function) and "x_1" (a terminal), we append "x_1" to the "$+$" and recursively apply `GrowTreeBasedAL`$(\cdot)$ with the second item (i.e., $[-, [*, \max]]$) in the adjacency list to grow the sub-tree since the function of the second item "$-$" is coincident with the function adjacent node in the first item. Since the adjacent nodes of "$-$" (i.e., "$*$" and "$\max$") are not included as items in the adjacency list, we randomly generate the sub-trees under "$-$".

3.2.2 Breeding Instructions Based on Adjacency Lists

In `CALX`$(\cdot)$, if the first parent $\mathbf{p}_1$ is an LGP individual, sub-tree and sub-DAGs are respectively selected based on parents' representation, a sub-tree for TGP parent and sub-DAGs for LGP parent (lines 14, 16, 19, and 21). Specifically, the sub-DAGs are selected by selecting an instruction segment from the LGP parent. Then, adjacency lists $\mathbf{L}_1$ and $\mathbf{L}_2$ are constructed respectively based on the selected sub-tree and sub-graphs (lines 15, 17, 20 and 22). A new instruction sequence is constructed and swapped into $\mathbf{p}_1$ to produce offspring by `GrowInstructionBasedAL`$(\cdot)$ (line 23).

CALX applies `GrowInstructionBasedAL`$(\cdot)$ to construct a new instruction segment for LGP, as shown in Alg. 4. First, $|\mathbf{L}_1|$ instructions are randomly removed (line 1). Then an insertion point s is selected for inserting the new instruction segment (line 2). Instructions are sequentially constructed based on $\mathbf{L}_2$ and swapped into the program context (lines 3-6). To connect the functions and maintain the topological structure of

Algorithm 4: `GrowInstructionBasedAL`

```
Input: An LGP individual c, adjacency list of the first parent $\mathbf{L}_1$, adjacency list of the second
       parent $\mathbf{L}_2$, crossover point $t_1$, instruction range $n_1$
Output: The LGP offspring c
1  Randomly remove $|\mathbf{L}_1|$ instructions from $[\mathbf{c}(t_1), \mathbf{c}(t_1 + n_1)]$;
2  $s \leftarrow t_1 + \texttt{randint}(n_1 - |\mathbf{L}_1|)$;
   // Construct an instruction list
3  for $j \leftarrow 1$ to $|\mathbf{L}_2|$ do
4      $[a, \mathbf{A}] \leftarrow \mathbf{L}_2(j)$;
5      $f \leftarrow$ randomly generate an instruction whose function is $a$;
       // Swap into the program context
6      Insert $f$ to $\mathbf{c}(s)$;
   // Assign registers
7  for $j \leftarrow s + |\mathbf{L}_2| - 1$ to $s$ do
8      $[a, \mathbf{A}] \leftarrow \mathbf{L}_2(s + \mathbf{L}_2 - j)$;
       // Assign destination registers
9      if $\mathbf{c}(j)$ is not effective to the final output then
10         Randomly mutate $\mathbf{c}(j)_d$ until $\mathbf{c}(j)$ is effective;
       // Assign source registers
11     for $g \leftarrow 1$ to $|\mathbf{A}|$ do
12         $b \leftarrow \mathbf{A}(g)$;
13         if $b$ is a function then
14             $\mathbf{L}' \leftarrow$ collect the entity indices from $[j, s]$ where $\mathbf{L}_2(k).fun = b$ and $k \in [j, s]$;
15             if $\mathbf{L}' \neq \emptyset$ then
16                 $\mathbf{c}(j)_{s,g} \leftarrow \mathbf{c}(\mathbf{L}'(\texttt{randint}(1, |\mathbf{L}'|)))_d$;
17             else
18                 if $j > 0$ and $\texttt{randint}(0, j) - 1 > 0$ then
19                     $\mathbf{c}(j)_{s,g} \leftarrow \mathbf{c}(\texttt{randint}(1, j))_d$;
20         else if $b$ is a constant then
21             $\mathbf{c}(j)_{s,g} \leftarrow b$;
22 Return c;
```

the functions based on $\mathbf{L}_2$, we check the instruction sequence reversely (i.e., from the top of the graph to the bottom) (lines 7-21) so that every newly generated instruction 1) is effective to the final output by altering the destination registers $\mathbf{c}(j)_d$ (lines 9-10), and 2) accepts the inputs from corresponding functions and constants based on $\mathbf{L}_2$ by altering the source registers $\mathbf{c}(j)_{s,g}$ (lines 11-21). Specifically, the effectiveness of an instruction is checked by an $\mathcal{O}(n)$ algorithm [5] (line 9). If the selected instruction is not effective, we randomly mutate the destination register of the instruction until it is effective. `GrowInstructionBasedAL`$(\cdot)$ assigns source registers based on the adjacent node b (line 12). If b is a function, we set the source register of the selected instruction $\mathbf{c}(j)$ as the destination register of a random instruction whose function is coincident with b (lines 14-16). If there is no such an instruction, we set the source register as the destination register of a random instruction precedent to $\mathbf{c}(j)$ (lines 18-19). The constant adjacent nodes replace the source registers directly (lines 20-21).

Fig. 5 shows an example of constructing an instruction list based on the adjacency list. First, we construct an instruction list in which the functions (i.e., “$+$” and “$-$”) are specified by the adjacency list. Note that the order of functions in the instruction list is reversed to the order in the adjacency list since LGP programs output final results from the bottom. Second, we swap the newly constructed instruction list into the program context. Third, we adjust the registers in the newly constructed instruction list to maintain the adjacency relationship in the offspring. In this example, we change the

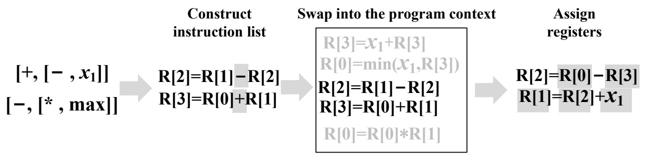

Figure 5: An example of constructing instructions by `GrowInstructionBasedAL`(·). Shadowed primitives are the focus of each step.

destination register R[3] into R[1] to ensure the new instruction list to be effective in the offspring, change R[0] in the second instruction into R[2] and change R[1] into x_1 to fulfill the adjacency relationship "$+ \rightarrow [-, x_1]$". To connect the newly constructed instruction list with the precedent instructions in existing programs, the source registers in the first instruction are also updated.

4 Empirical Studies of CRGP-TL

We verify the effectiveness of CRGP-TL by two applications, symbolic regression and automatic design of decision rules in dynamic combinatorial optimization problems. Symbolic regression is a supervised learning problem, in which GP learns regression models to map the input features to given target outputs without presuming the model structure. GP has shown great success in solving symbolic regression problems [3, 28]. Automatically designing decision rules for dynamic combinatorial optimization problems uses GP to automatically learn decision rules to make instant reactions for dynamic events in combinatorial optimization problems. Unlike symbolic regression problems where there are target outputs for training, the decision rule design problems have no target outputs available. GP methods have to search for solutions based on a black-box performance indicator (e.g., the performance of simulations). Specifically, we focus on dynamic job shop scheduling (DJSS) as an example of dynamic combinatorial optimization in this paper. DJSS is a common and important combinatorial optimization problem in real-world practice. Designing instant decision rules for DJSS is a challenging problem for all GP representations [11, 23].

4.1 Comparison Design

To verify the effectiveness of CRGP-TL, we compare CRGP-TL with three baseline methods in these two applications. The first two are the basic TGP and LGP. Then, a baseline GP method with two independent sub-populations is developed (denoted as "TLGP"). The two sub-populations independently evolve tree-based and linear representations by the basic genetic operators for each representation and do not exchange genetic materials among representations. The best individual between the two sub-populations is output as the final solution. To ensure fairness, we set the parameters of the compared GP methods for the two applications respectively, following the settings used in existing studies [8, 18, 26]. The parameter settings for the two applications are demonstrated in Sections 4.2.2 and 4.3.2.

4.2 Application I: Symbolic Regression

4.2.1 Problem Description

In this section, we apply CRGP-TL to symbolic regression problems. We select three synthetic benchmarks and five real-world benchmarks, as shown in Table 1. The bench-

Table 1: The symbolic regression problems

Benchmarks	Function	#Features	Data range	#Points (Train,Test)
Synthetic benchmarks				
Nguyen4	$f(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x$	1	[-1,1]	(20,1000)
Keijzer11	$f(x,y) = xy + \sin((x-1)(y-1))$	2	[-1,1]	(100,900)
R1	$f(x) = \frac{(x+1)^3}{x^2-x+1}$	1	[-2,2]	(20,1000)
Real-world benchmarks				
Airfoil	unknown	5	-	(1127,376)
BHouse	unknown	13	-	(380,126)
Tower	unknown	25	-	(3749,1250)
Concrete	unknown	8	-	(772,258)
Redwine	unknown	11	-	(1199, 400)

marks are selected from recently published papers for solving symbolic regression [1, 22]. The ground truth functions of the synthetic benchmarks cover a wide range of functions (e.g., $\times$ and $\sin$), and the real-world benchmarks have various numbers of features and data ranges.

This paper applies relative square error (RSE) to measure the performance of GP methods, as shown in Eq. 1.

$$\text{RSE} = \frac{\text{MSE}(\mathbf{y}, \hat{\mathbf{y}})}{\text{VAR}(\mathbf{y})} = \frac{\sum_i^n (y_i - \hat{y_i})^2}{\sum_i^n (y_i - \overline{y})^2} \tag{1}$$

where MSE is the mean square error, VAR is the variance, and $\mathbf{y}$ and $\hat{\mathbf{y}}$ are the target output and estimated output respectively. $\overline{y}$ is the average of the target output. A small RSE value implies that a regression model has a good fitting performance with the given data.

4.2.2 Parameter Settings

In symbolic regression problems, LGP evolves a population with 256 individuals for 200 generations. LGP applies linear crossover, effective macro mutation, effective micro mutation, and reproduction in breeding [5], with a genetic operator rate of 30%:30%:30%:10% respectively. Each LGP individual has at most 100 instructions and manipulates 8 registers. TGP evolves a population with 1024 individuals for 50 generations, and applies crossover, mutation, and reproduction in breeding, with a genetic operator rate of 80%:15%:5% respectively. Each TGP individual has a maximum tree depth of 10.

For the two algorithms with multiple representations (i.e., TLGP and CRGP-TL), each sub-population has 128 individuals and evolves for 200 generations. The parameters of the CRGP-TL are defined based on the baseline method. Specifically, the knowledge transfer rate is defined as 30% by default, without loss of generality. Since the proposed adjacency list-based operators used to transfer knowledge among sub-populations can also exchange the genetic materials for the same representation, the LGP sub-population in CRGP-TL does not apply linear crossover operator, and the TGP sub-population in CRGP-TL reduces the crossover rate from 80% to 50%. All the compared methods apply a tournament selection with a size of 7 to perform parent selection and apply an elitism selection with an elitism rate of 10% to retain elite individuals. The other parameters of TLGP and CRGP-TL are kept the same as in the basic TGP and LGP methods.

All the compared GP methods use the same function set and terminal set (LGP methods have registers in the terminal set additionally). The function set includes 8 functions, which are $\{+, -, \times, \div, \sin, \cos, \ln(|\cdot|), \sqrt{|\cdot|}\}$[3]. The input feature set is defined based on the inputs of benchmark problems.

4.3 Application II: Dynamic Job Shop Scheduling Problems

4.3.1 Problem Description

This section applies CRGP-TL to design decision rules (i.e., dispatching rules) for DJSS [31, 49]. We focus on the DJSS problems with new job arrival, in which jobs come into the job shop over time. A DJSS problem has a set of jobs $\mathcal{J}$ and a set of machines $\mathcal{M}$. Each $j \in \mathcal{J}$ consists of a sequence of operations $\mathbf{O}_j = \{o_{j1}, o_{j2}, ..., o_{jl_j}\}$ where l_j is the number of operations in job j. Every o_{ji} can only be processed after $o_{j,i-1}$ is finished ($2 \le i \le l_j$). Each job j has an arrival time α_j, a due date d_j, and a weight ω_j. $o_{ji}(1 \le i \le l_j)$ is going to be processed by a specific machine $\pi(o_{ji}) \in \mathcal{M}$ with a positive processing time $\delta(o_{ji})$. Every machine can only process one operation at any time, and the execution of the operation cannot be interrupted by other operations.

A DJSS simulation is built up based on the description, and the settings of the DJSS simulations are designed based on the existing literature [21]. Specifically, there are 10 machines in the job shop. Each job has 2 to 10 operations, each operation with a processing time ranging from 1 to 99. To evaluate the performance of GP individuals in a steady job shop, we warm up the job shop with the first 1000 jobs and only take the subsequent 5000 jobs into account when evaluating GP individuals.

Jobs come into the job shop based on a Poisson distribution, as shown in Eq. 2. t_a is the time interval before the next job arrival. λ is the mean processing time of a job in the job shop, defined by Eq. 3. ν is the average number of operations in the jobs, and μ is the average processing time of operations. The utilization level of machines ρ defines the arrival rate of jobs. A large ρ implies that jobs will be processed by the job shop very quickly (i.e., a small mean actual processing time of jobs) and that new jobs arrive to the job shop in a shorter time.

$$P(t_a = \text{time interval before the next job arrival time}) \sim \exp(-\frac{t_a}{\lambda}) \tag{2}$$

$$\lambda = \frac{\nu \cdot \mu}{\rho \cdot |\mathcal{M}|} \tag{3}$$

The simulation performance is seen as the performance of GP individuals. There are six optimization objectives for the job shop in our work, which are formulated as follows. T_{max} and F_{max} denote the maximum tardiness (tardiness of job j: $T_j = \max(c_j - d_j, 0), j \in \mathcal{J}$) and flowtime (flowtime of job j: $F_j = c_j - a_j, j \in \mathcal{J}$) among all the jobs respectively. c_j, d_j, and a_j denote the completion time, the due date, and the arrival time of job j. T_{mean} and F_{mean} denote the mean tardiness and flowtime over all the jobs respectively. WT_{mean} and WF_{mean} denote the weighted mean tardiness and flowtime respectively. The maximum and mean objectives comprehensively indicate the worst case and average performance of the compared methods.

1. $T_{max} = \max_{j \in \mathcal{J}}(\max(c_j - d_j, 0))$
2. $T_{mean} = \frac{\sum_{j \in \mathcal{J}}(\max(c_j - d_j, 0))}{|\mathcal{J}|}$
3. $WT_{mean} = \frac{\sum_{j \in \mathcal{J}}(\omega_j \times \max(c_j - d_j, 0))}{|\mathcal{J}|}$

[3] $\div$ returns 1.0 if the denominator equals 0.0. $\ln(|\cdot|)$ returns the operand if the raw output is smaller than -50.

Table 2: The terminal set

Notation	Description	Notation	Description
PT	Processing time of an operation in a job	W	Weight of a job
NPT	Processing time of the next operation for a certain operation in a job	rDD	Difference between the given due date of a job and the system time
WINQ	Total processing time of the operations in a machine buffer. The machine is the corresponding machine for the next operation in a job	NWT	Waiting time of the next to-be-ready machine
WKR	Total remaining processing time of a job	TIS	Difference between system time and the job arrival time
rFDD	Difference between the given due date of an operation and the system time	SL	Slack: difference between the given due date and the sum of the system time and WKR
OWT	Waiting time of an operation	NIQ	Number of operations in a machine buffer
NOR	Number of remaining operations of a job	WIQ	Total processing time of operations in a machine buffer
NINQ	Number of operations in the buffer of a machine which is the corresponding machine of the next operation in a job	MWT	Waiting time of a machine

4. $F_{max} = \max_{j \in \mathcal{J}}(c_j - a_j)$

5. $F_{mean} = \frac{\sum_{j \in \mathcal{J}}(c_j - a_j)}{|\mathcal{J}|}$

6. $WF_{mean} = \frac{\sum_{j \in \mathcal{J}} \omega_j (c_j - a_j)}{|\mathcal{J}|}$

To comprehensively verify the performance of the proposed method on different difficulty levels, two utilization levels (i.e., 0.85 and 0.95) are adopted for the simulation. A higher utilization level implies a busier job shop and more difficulty to find a schedule. In short, this paper tests twelve scenarios which are notated by "⟨Objective, Utilization level⟩". The twelve scenarios are $\langle T_{max}, 0.85\rangle$, $\langle T_{max}, 0.95\rangle$, $\langle T_{mean}, 0.85\rangle$, $\langle T_{mean}, 0.95\rangle$, $\langle WT_{mean}, 0.85\rangle$, $\langle WT_{mean}, 0.95\rangle$, $\langle F_{max}, 0.85\rangle$, $\langle F_{max}, 0.95\rangle$, $\langle F_{mean}, 0.85\rangle$, $\langle F_{mean}, 0.95\rangle$, $\langle WF_{mean}, 0.85\rangle$, and $\langle WF_{mean}, 0.95\rangle$.

4.3.2 Parameter Settings

The parameters of the compared methods are set based on the common settings in using GP methods to design dispatching rules for DJSS problems [18, 49]. Specifically, all the TGP individuals have a maximal tree depth of eight, and all the LGP individuals have at least one instruction and at most fifty instructions. The knowledge transfer rate between TGP and LGP populations is the same as in symbolic regression problems (i.e., 30%). The rest of the parameters are kept the same as in Section 4.2.2. In DJSS problems, all the compared methods adopt the function set (i.e., $\{+, -, \times, \div, \max, \min\}$) and the terminal set in Table 2.

For each independent run, a GP method first evolves on the training set and produces a final output rule based on a validation set with 10 DJSS instances. The rule with the best performance on the validation set is tested on 50 unseen DJSS instances. The test performance is defined as the mean performance on these test instances. The GP methods are trained on one DJSS instance for each generation, and the DJSS training instances are rotated every generation to improve the generalization ability of GP rules

Evolutionary Computation Just Accepted MS.
https://doi.org/10.1162/evco_a_00025
© 2025 by the Massachusetts Institute of Technology

[17]. All the compared methods have the same maximum number of simulations (i.e., fitness evaluation) in training.

4.4 Empirical Results

This section analyzes the test and training performance of the compared methods for solving the symbolic regression and DJSS problems.

4.4.1 Test Performance

Table 3 shows the average test performance of the compared methods in solving the two kinds of problems. We perform a Friedman test ($\alpha = 0.05$) with a Bonferroni correction on the test performance of the compared methods. We treat the rank of the median over 50 independent runs on a dataset as a sampled value of the compared methods in the Friedman test.

The p-value of the Friedman test is 7.4E-4, which indicates a significant difference among the compared methods. Moreover, CRGP-TL has the best (i.e., smallest) mean rank of test performance among all the compared methods, with very promising pair-wise comparison p-values with other compared methods. The results and statistical analyses confirm that the proposed CRGP-TL has a significantly better overall performance than the other three compared methods.

To further investigate the effectiveness of the compared methods on different datasets, Table 3 shows the results of Wilcoxon rank-sum test with Bonferroni correction and an α of 0.05 over the test performance of the compared methods. $+$, $-$, and $\approx$ denote that a certain compared method is significantly better than, worse than, or performs similarly to the proposed CRGP-TL respectively, based on the Wilcoxon rank-sum test. We treat the test performance of an independent run as a sampled value in the Wilcoxon rank-sum test. The best mean performance is highlighted in bold font. We see that in most datasets and scenarios, CRGP-TL has a very competitive performance with the compared methods. More specifically, CRGP-TL has significantly better test performance than the compared methods on at least 7 of 20 datasets and scenarios based on the Wilcoxon rank-sum test. The results confirm that sharing knowledge between tree-based and linear representation successfully improves the effectiveness of GP methods.

4.4.2 Training Performance

To analyze the learning ability of the compared methods, Fig. 6 shows the test performance of the compared methods over generations in eight example problems. Specifically, we select four real-world symbolic regression benchmarks and four DJSS scenarios with a high utilization level (i.e., 0.95) as the example problems since the real-world symbolic regression benchmarks and the DJSS scenarios with a high utilization level have better real-world practical value.

CRGP-TL (i.e., red curves) has better test performance (i.e., lower objective values) within fewer fitness evaluations than other methods in many cases, such as BHouse and Concrete. In some other cases, though CRGP-TL levels off at a test performance similar to the other compared methods, it achieves the test performance earlier than the compared methods at the early stage of the evolution. The results imply that CRGP-TL has a very competitive training performance with other compared methods in both symbolic regression and DJSS problems and can find solutions with better effectiveness within fewer fitness evaluations in some specific cases.

Table 3: The mean test performance (std.) of the compared methods

Datasets or scenarios	TLGP	TGP	LGP	CRGP-TL
Test RSE (std.) of Symbolic Regression				
Nguyen4	0.069 (0.059) ≈	0.053 (0.091) ≈	0.149 (0.248) –	**0.049 (0.041)**
Keijzer11	1.744 (9.749) –	0.273 (0.121) –	0.339 (0.142) –	**0.217 (0.121)**
R1	0.035 (0.029) –	0.022 (0.023) –	0.034 (0.035) –	**0.011 (0.017)**
Airfoil	0.667 (0.091) –	0.638 (0.117) –	0.643 (0.132) –	**0.526 (0.1)**
Bhouse	0.384 (0.1) ≈	0.392 (0.131) ≈	0.404 (0.126) ≈	**0.362 (0.104)**
Tower	0.358 (0.052) –	0.364 (0.053) –	0.345 (0.046) –	**0.316 (0.031)**
Concrete	0.496 (0.096) –	0.438 (0.107) –	0.471 (0.099) –	**0.291 (0.063)**
Redwine	0.745 (0.042) –	0.761 (0.036) –	0.759 (0.034) –	**0.724 (0.031)**
Test Objective Values (std.) of DJSS				
⟨Tmax,0.85⟩	1939.8 (50.4) ≈	**1928.4 (40.4)** ≈	1956.3 (53.8) –	1931.8 (58.3)
⟨Tmax,0.95⟩	4009.2 (98.9) ≈	4060.6 (116) –	3999.2 (90.9) ≈	**3968.5 (81.8)**
⟨Tmean,0.85⟩	417 (3.2) ≈	417.3 (2.5) ≈	417.9 (2.3) –	**416.2 (2.5)**
⟨Tmean,0.95⟩	1116.3 (9.3) ≈	**1116.2 (10)** ≈	1118.2 (10.7) ≈	1117.7 (11)
⟨WTmean,0.85⟩	725.8 (6.1) ≈	727.5 (6.5) –	724.3 (5.4) ≈	**723.4 (5.9)**
⟨WTmean,0.95⟩	1730.4 (23.6) ≈	1747.4 (29.6) –	1729.6 (27.7) ≈	**1727 (24.2)**
⟨Fmax,0.85⟩	2506.6 (50.3) –	2494.3 (30) ≈	2509.8 (58.8) –	**2485 (29.5)**
⟨Fmax,0.95⟩	4544.3 (98.5) ≈	4572.3 (96.5) –	4585.4 (126.1) –	**4534.3 (112)**
⟨Fmean,0.85⟩	864 (3.2) ≈	863.2 (4.2) ≈	864.7 (4) –	**862.8 (3)**
⟨Fmean,0.95⟩	1564.9 (10.3) ≈	1565.4 (8.5) ≈	1566.8 (10.8) ≈	**1563.6 (10.4)**
⟨WFmean,0.85⟩	1704 (10.2) ≈	1705.4 (7.5) ≈	1702.6 (7) ≈	**1702.6 (6.2)**
⟨WFmean,0.95⟩	2718.4 (26.4) ≈	2730.1 (29.3) –	2715.8 (16.4) ≈	**2712.8 (17.8)**
win-draw-lose	0-13-7	0-9-11	0-8-12	
Mean rank	2.9	2.625	3.2	**1.275**
p-value (vs. CRGP-TL)	3E-4	5E-3	1E-5	

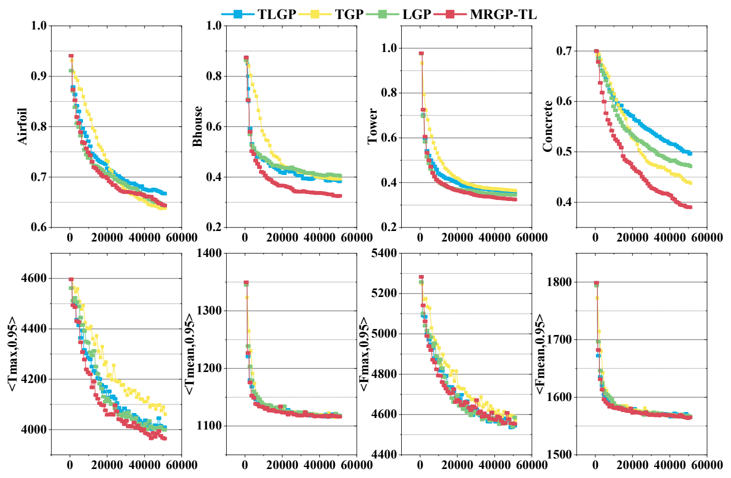

Figure 6: Test performance of the compared methods over generations in the nine symbolic regression benchmarks. X-axis: fitness evaluations. Y-axis: average test RSE for symbolic regression problems and average test objective values for DJSS problems.

Evolutionary Computation Just Accepted MS.
https://doi.org/10.1162/evco_a_00025
© 2025 by the Massachusetts Institute of Technology

4.5 Summary

In summary, CRGP-TL substantially improves the effectiveness of baseline methods and has a very competitive training performance with other compared methods in solving symbolic regression and DJSS problems. The results imply that sharing search information among different GP representations is a very potential direction in improving GP performance. The experiments on tree-based and linear representations shows that CRGP-TL can automatically take advantage of the most suitable representation to achieve better performance, which is less dependent on the domain knowledge of the suitable representations for different problems than GP methods with a single representation. Further comparison between CRGP-TL and TLGP confirms that the performance gain of CRGP stems from the knowledge sharing among different representations, which is fulfilled by the newly proposed cross-representation adjacency list-based crossover.

5 Further Analyses and Discussion

To have a further understanding of the cross-representation mechanism, this section conducts six investigations based on the two applications. First, we compare CRGP-TL with two state-of-the-art GP methods in solving symbolic regression and DJSS problems respectively to further verify the effectiveness of CRGP-TL. Second, we analyze the average program size of the GP population over generations. Third, we analyze the sensitivity of the key parameters in CRGP-TL. Fourth, we highlight the benefit of knowledge sharing between representations by comparing CRGP-TL with other diversification methods. We also investigate the effectiveness of different computation resource allocations to GP representations. Finally, we study the effectiveness of knowledge sharing between tree-based and linear representations and take two examples of adjacency lists to analyze the knowledge sharing in CRGP.

5.1 Comparison with Advanced Methods

We compare CRGP-TL with two recently published GP methods, semantic linear genetic programming [22] for symbolic regression problems and grammar-guided linear genetic programming [19] for DJSS problems. These two compared methods have been published recently and have shown promising performance in symbolic regression and DJSS problems respectively. The experiment settings in this section follow the ones in [22] and [19] for a fair comparison. To further investigate the performance impact of the cross-representation mechanism, we developed two algorithms that incorporate the cross-representation mechanism with the two advanced methods, respectively. For the sake of simplicity, we add the advanced GP ([22] for symbolic regression problems and [19] for DJSS problems) as the third sub-population besides basic TGP and LGP. The three sub-populations with different representations simultaneously evolve based on the proposed mechanism. We simply denote semantic linear genetic programming and grammar-guided linear genetic programming as ADVAN for symbolic regression and DJSS problems and denote both the advanced methods with the cross-representation mechanism as CRGP-advan.

Table 4 applies the Friedman test with the Bonferroni correction and the Wilcoxon rank-sum test with $\alpha = 0.05$ to analyze the results where $+$, $-$, and $\approx$ denote that a certain compared method is significantly better than, worse than, or performs similarly to ADVAN respectively, based on the Wilcoxon rank-sum test.

For symbolic regression problems, CRGP-TL is significantly worse than ADVAN in all the datasets based on the Wilcoxon rank-sum test. Although CRGP-advan has a

Table 4: Average Test performance (std.) comparison among CRGP-TL and advanced methods

Datasets or scenarios		ADVAN	CRGP-TL	CRGP-advan
Symbolic Regression	Nguyen4	**0.001 (0.001)**	0.041 (0.05) −	0.009 (0.019) −
	Keijzer11	**0.031 (0.012)**	0.194 (0.128) −	0.047 (0.033) ≈
	R1	**0.001 (0.002)**	0.008 (0.009) −	0.003 (0.004) −
	Airfoil	0.402 (0.022)	0.489 (0.111) −	**0.391 (0.046)** ≈
	Bhouse	**0.212 (0.028)**	0.326 (0.102) −	0.223 (0.028) −
	Tower	**0.137 (0.012)**	0.302 (0.034) −	0.144 (0.016) −
	Concrete	**0.187 (0.014)**	0.272 (0.05) −	0.211 (0.027) −
	Redwine	**0.651 (0.014)**	0.717 (0.035) −	0.662 (0.025) ≈
	mean rank	**1.125**	3	1.875
	p-value(vs. ADVAN)		5E-4	0.401
DJSS	⟨Tmax,0.85⟩	1922.1 (42.9)	1931.8 (58.3) ≈	**1920.3 (35.3)** ≈
	⟨Tmax,0.95⟩	**3943.1 (84)**	3968.5 (81.8) ≈	3992.4 (121) ≈
	⟨Tmean,0.85⟩	417.7 (2.6)	**416.2 (2.5)** ≈	417.8 (2.8) ≈
	⟨Tmean,0.95⟩	**1116.7 (8.7)**	1117.7 (11) ≈	1122.9 (15) ≈
	⟨WTmean,0.85⟩	723.6 (7.5)	**723.4 (5.9)** ≈	726.2 (7.2) ≈
	⟨WTmean,0.95⟩	**1724.4 (26.6)**	1727 (24.2) ≈	1732.3 (26.4) ≈
	⟨Fmax,0.85⟩	2534.6 (74.1)	**2485 (29.5)** +	2488.2 (36.2) +
	⟨Fmax,0.95⟩	4599.7 (80.6)	**4534.3 (112)** +	4551.4 (97.6) +
	⟨Fmean,0.85⟩	864.6 (3.2)	**862.8 (3)** +	865.2 (4) ≈
	⟨Fmean,0.95⟩	1565.3 (10.9)	**1563.6 (10.4)** ≈	1567.8 (14.4) ≈
	⟨WFmean,0.85⟩	1701.7 (6.1)	1702.6 (6.2) ≈	**1701.2 (4.8)** ≈
	⟨WFmean,0.95⟩	2722.8 (25.4)	**2712.8 (17.8)** ≈	2713.7 (23.1) ≈
	mean rank	**1.833**	**1.833**	2.333
	p-value(vs. ADVAN)		1	0.633

statistically similar overall performance to ADVAN (p-value=0.401), it is less competitive than ADVAN in some datasets.

For DJSS problems, CRGP-TL has a very competitive performance with ADVAN. CRGP-TL has better test performance on three scenarios than ADVAN and has the same mean rank based on Friedman's test. CRGP-TL also has the best mean performance on seven of twelve DJSS problems. Cooperating CRGP with the advanced GP method also shows a competitive performance.

Based on these results, we conclude that CRGP has potential in enhancing the performance of existing GP representations. Based on the basic tree-based and linear-based representations, CRGP-TL shows very competitive results with manually designed advanced GP methods in DJSS problems. However, its inferior performance compared to advanced GP methods in symbolic regression implies that the effectiveness of CRGP-TL is limited by the less effective representations since they might waste too many computation resources. The results imply that to guarantee the effectiveness of the cross-representation mechanism, we should incorporate similarly competitive representations in case some less effective representations consume a large amount of computation resources.

5.2 Program Size

To further understand the evolution of CRGP-TL, we analyze the average program size of the population in all the compared methods for solving eight example problems, as shown in Fig. 7. Specifically, we show the program size of tree-based and linear programs respectively in TLGP and CRGP-TL, denoted by "-T" and "-L" (e.g., tree-based programs in TLGP are denoted as "TLGP-T"). We use the number of tree nodes to denote the program size of tree-based programs and use the number of effective instructions multiplied by a factor of 2.0 to denote the program size of linear

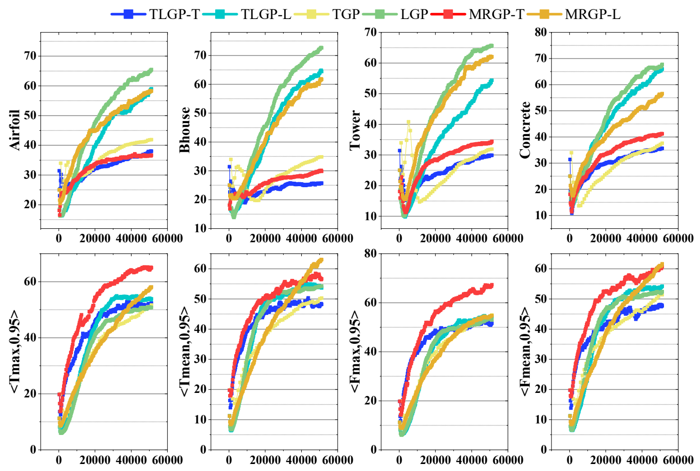

Figure 7: The average program size of the population from the compared methods over generations over 50 independent runs. X-axis: fitness evaluations, Y-axis: the average program size of the population.

programs[21]. We can see that the average program size from the same representation grows similarly in all the tested problems. For example, in the four symbolic regression benchmarks, TLGP-L, LGP, and CRGP-L all grow from about 20 to about 65, and TLGP-T, TGP, and CRGP-T all grow from about 10 to about 25. The similar growing curves of the same representation confirm that the proposed cross-representation adjacency list-based crossover operator has a similar variation step size with basic genetic operators and does not significantly change the average program size of the population. Fully utilizing the interplay between tree-based and linear representations improves the effectiveness of solutions without enlarging the program size of the solutions.

5.3 Parameter Sensitivity Analyses

The knowledge transfer rate among representations θ_t is a newly introduced parameter. To investigate the influence of θ_t on performance, CRGP-TL with different transfer rates are compared in this section. Specifically, we investigate the performance of CRGP-TL with a θ_t of 0%, 10%, 30%, 50%, and 70% respectively, which are denoted as TL0 (i.e., TLGP), TL10, TL30, TL50, and TL70.

The test performances of CRGP-TL with different θ_ts are shown in Fig. 8. We see that CRGP-TL methods with $\theta_t > 0$ on average have smaller (i.e., better) objective values than CRGP without any knowledge sharing (i.e., TL0 or TLGP). Besides, TL10, TL30, TL50, and TL70 have statistically similar test performance in most cases. But in some cases such as Airfoil, Concrete, $\langle T_{max}, 0.95\rangle$, and $\langle F_{max}, 0.95\rangle$, the increase of θ value improves the performance of CRGP-TL on average. To conclude, θ_t, the knowledge transfer rate among representations shows robust performance in principle, but tuning on specific scenarios has the potential to further improve CRGP-TL performance.

5.4 Benefit of Cross-representation Knowledge Sharing

Sharing knowledge between tree-based and linear representations helps GP methods discover more diverse and effective building blocks and jump out of local optima.

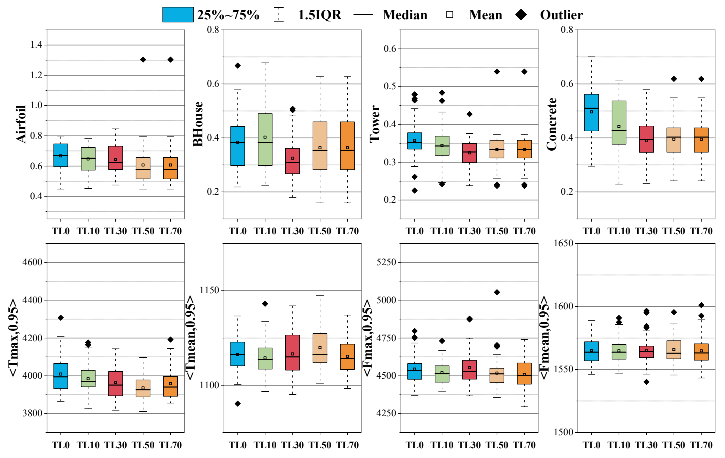

Figure 8: The box plots on the test performance of CRGP-TL with different θ_t values over 50 independent runs.

Table 5: The average test performance (std.) of exchanging search information by basic crossover operators and CALX

Datasets and scenarios	CRGP-TL	CRGP-rand	MP-TGP	MP-LGP
Airfoil	0.526 (0.1)	0.526 (0.106) $\approx$	**0.51 (0.071)** $\approx$	0.579 (0.116) $\approx$
Bhouse	0.362 (0.104)	0.376 (0.124) $\approx$	**0.353 (0.104)** $\approx$	0.418 (0.102) $-$
Tower	0.316 (0.031)	0.316 (0.03) $\approx$	0.32 (0.041) $\approx$	**0.311 (0.043)** $\approx$
Concrete	**0.291 (0.063)**	0.332 (0.078) $-$	0.366 (0.089) $-$	0.334 (0.103) $\approx$
$\langle$Tmean,0.85$\rangle$	**416.2 (2.5)**	418 (3.5) $-$	417.7 (3.2) $\approx$	418.3 (2.6) $-$
$\langle$Tmean,0.95$\rangle$	1117.7 (11)	1122 (11.8) $\approx$	1119.1 (11.4) $\approx$	**1117.3 (9.5)** $\approx$
$\langle$WTmean,0.85$\rangle$	**723.4 (5.9)**	727.8 (6.6) $-$	730 (8.9) $-$	724.3 (6.2) $\approx$
$\langle$WTmean,0.95$\rangle$	**1727 (24.2)**	1739.2 (22.9) $-$	1741.6 (31.3) $\approx$	1731.9 (22.5) $\approx$
$\langle$WFmean,0.85$\rangle$	1702.6 (6.2)	1705.5 (7.3) $\approx$	1706.2 (6.4) $-$	**1701.7 (6.5)** $\approx$
$\langle$WFmean,0.95$\rangle$	**2712.8 (17.8)**	2720.4 (28.7) $\approx$	2727.9 (30.7) $\approx$	2715.6 (21.5) $\approx$
mean rank	**1.4**	2.75	2.95	2.9
p-values (vs. CRGP-TL)		0.113	0.042	0.054

To verify the effectiveness of knowledge sharing between the two representations, this section compares CRGP-TL with other diversification methods. Specifically, we implement three multipopulation GP methods, CRGP-rand, MP-TGP, and MP-LGP. CRGP-rand evolves two sub-populations, each with tree-based or linear representations. However, CRGP-rand exchanges random adjacency lists to make the population more diverse. MP-TGP and MP-LGP evolve two subpopulations of tree-based and linear-based representations, respectively. These subpopulations in MP-TGP and MP-LGP apply genetic operators with different settings to improve the diversity of subpopulations. CRGP-rand, MP-TGP, and MP-LGP exchange building blocks with the same exchange rate as CRGP-TL (i.e., $\theta_t = 30\%$).

Table 5 shows the test performance of the four compared methods for solving ten example problems. We apply the Friedman test ($\alpha = 0.05$) with a Bonferroni correction to analyze the overall performance. The p-value of the Friedman test is 0.019, indicating a significant difference in the test performance of the compared methods. The mean ranks given by the Friedman test verify that CRGP-TL has the best test performance among the four compared methods. Specifically, the p-values of the pair-wise compar-

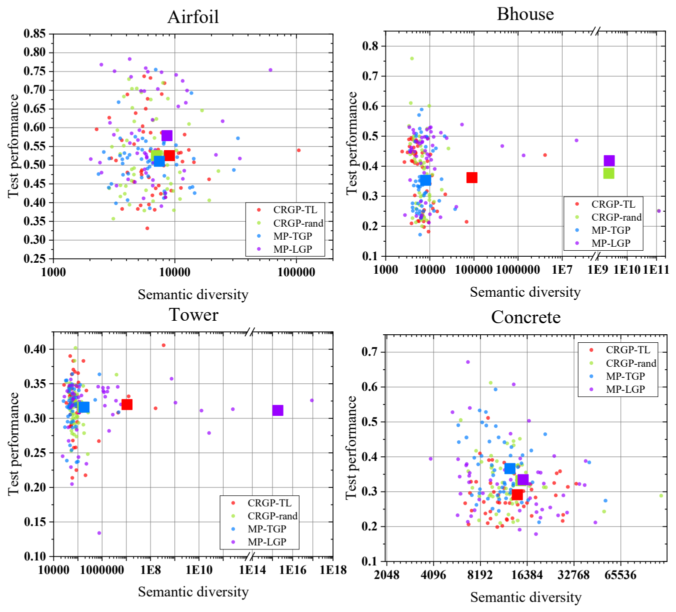

Figure 9: Test performance (Y-axis) over semantic diversity (X-axis) on the four example problems. Dots are the test performance and semantic diversity of an independent run, and squares are the average of test performance and semantic diversity for each method.

ison show that CRGP-TL is significantly better than MP-TGP. Based on the Friedman test, we apply the Wilcoxon rank-sum test ($\alpha = 0.05$) with a Bonferroni correction to analyze the performance of each dataset. Although the overall performance of CRGP-TL is not significantly different from MP-LGP and CRGP-rand, the post-hoc Wilcoxon rank-sum test verifies that CRGP-TL has superior performance to MP-LGP and CRGP-rand on two problems and has better mean performance than MP-LGP and CRGP-rand on seven of the ten problems. The results confirm that knowledge sharing between representations is more effective than simple diversification methods in enhancing GP performance.

To further verify that the cross-representation knowledge sharing helps GP find different but effective building blocks, we investigate the semantic diversity of elite individuals (i.e., top 20 individuals) at the final generation. We define the Euclidean distance of program outputs over all training instances as semantic differences and the average pair-wise semantic differences over elite individuals as semantic diversity. We take the four real-world symbolic regression problems as examples. Fig. 9 shows the scatter plots of the test performance over semantic diversity over 50 independent runs. The squares are the centroids of the scatter dots of the four compared methods. We can see that our proposed CRGP-TL (i.e., the red points) achieves better semantic diversity than MP-TGP and CRGP-rand in three problems. Although MP-LGP has better semantic diversity than CRGP-TL in most cases, MP-LGP often sacrifices its effectiveness (also as shown in Table 5). The results confirm that CRGP-TL finds more diverse and effective building blocks.

To verify that the cross-representation knowledge sharing helps GP jump out of

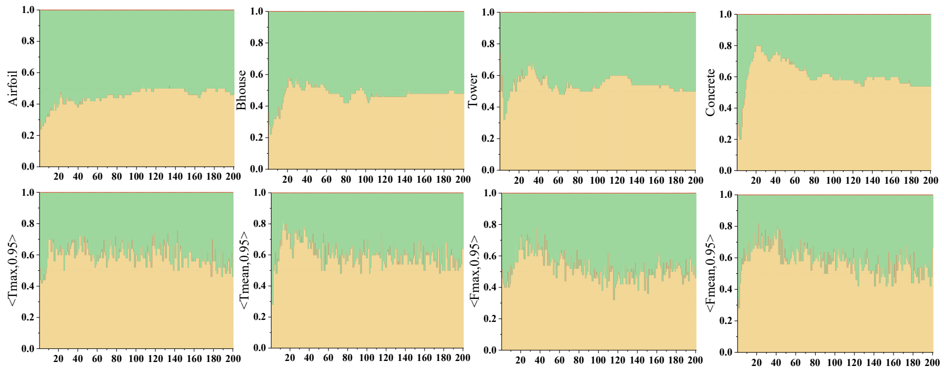

Figure 10: The average ratio of tree-based and linear representations producing the best-of-run individual over generations in CRGP-TL. X-axis: generations, Y-axis: ratio of producing the best-of-run individual. The green (i.e., upper) area denotes the ratio of linear representation, and the yellow (i.e., lower) area denotes the ratio of tree-based representation.

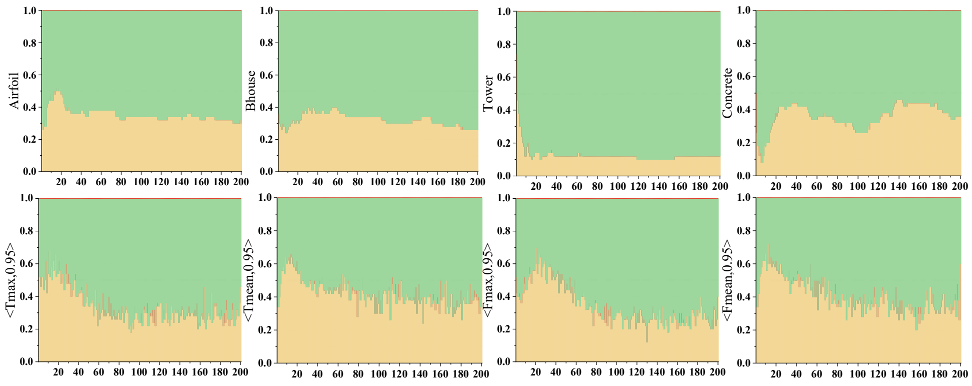

Figure 11: The average ratio of tree-based and linear representations producing the best-of-run individual over generations in TLGP (i.e., **without knowledge sharing**). X-axis: generations, Y-axis: ratio of producing the best-of-run individual.

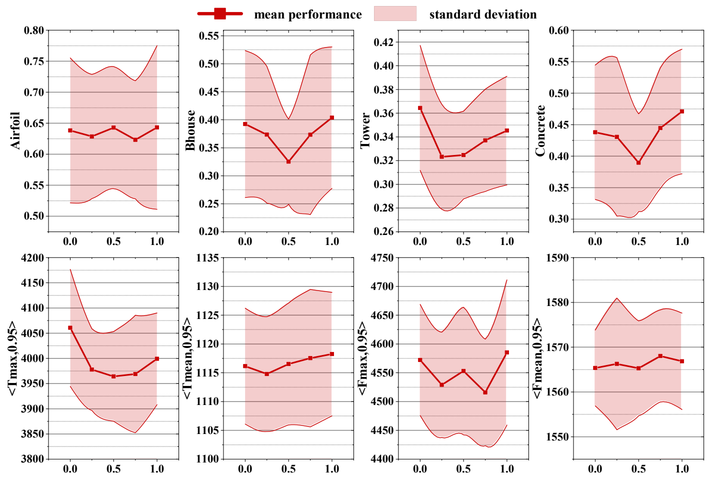

Figure 12: Test performance of different population ratios in CRGP-TL. X-axis: LGP population proportion. Y-axis: test performance of CRGP-TL

local optima, this section investigates the average ratio of each GP representation producing the best-of-run individuals over generations. Fig. 10 shows the average ratio in CRGP-TL. By contrast, Fig. 11 shows the ratio without knowledge sharing (i.e., tree-based and linear representations in TLGP). In CRGP (i.e., Fig. 10), tree-based and linear representations help each other to find effective individuals with a similar ratio (i.e., 0.4∼0.6) in all the selected problems. For example, in the Concrete dataset, LGP has better solutions at the beginning of evolution. Then, tree-based representation rapidly finds more effective solutions with the help of LGP search information from generations 10 to 30. After 30 generations, effective solutions in the tree-based representation in turn help the linear representation find effective solutions and catch up with the tree-based representation at about generation 80. On the contrary, in TLGP for the Concrete dataset, the best-of-run individuals are mainly produced by the linear representation during most of the evolution (i.e., the green area covers over 60% at each generation).

5.5 Representations with Various Computation Budgets

Different problems often have their own suitable GP representations, implying that allocating different amounts of computation resources to different representations in CRGP-TL might bring benefits to the performance of CRGP-TL. To investigate the impact of computation budgets on different GP representations, we adjust the allocation of computation resources by increasing the LGP population proportion from 0% to 100% (and decreasing the TGP population proportion from 100% to 0%). Specifically, we investigate five settings of LGP proportions, which are 0%, 25%, 50%, 75%, and 100%. The average test performance and standard deviation of CRGP-TL are shown in Fig. 12.

In most of the eight tested problems, the mean test performances in the same problem rougly form a "V" shape over different population ratios. This implies that CRGP-TL achieves a relatively good mean test performance and standard deviation when LGP and TGP share a similar proportion of computation resources (i.e., similarly large sub-populations). Although the performance of CRGP-TL can be further improved by

$\mathbf{L}_{\mathrm{TGP}}$ = ([×, [max, PT]] [max, [×, +]] [×, [WIQ, +]] [+, [+, PT]] [+, [+, −]] [+, [×, PT]] [×, [NPT, NINQ]] [−, [NOR, NPT]] [+, [+, min]] [+, [×, PT]] [×, [NPT, NINQ]] [min, [÷, +]] [÷, [max, −]] [max, [max, min]] [max, [NWT, NINQ]] [min, [rFDD, PT]] [−, [max,×]] [max, [WIQ, WKR]] [×, [OWT, WIQ]] [+, [WKR, rFDD]])

$\mathbf{L}_{\mathrm{LGP}}$ = ([×, [min, max]] [min, [−, −]] [max, [−, −]] [−, [max, min]] [min, [min, min]] [min, [max, +]] [max, [max, max]] [max, [−, −]] [−, [max, max]] [−, [×,×]] [×, [max, NINQ]] [max, [×, NPT]] [×, [max, PT]] [max, [+,+]] [min, [÷, ÷]] [+, [÷, +]] [+, [+, ÷]] [÷, [÷, NINQ]] [÷, [×, −]] [×, [÷, min]] [÷, [÷, min]] [max, [+, max]] [max, [min,×]] [×, [PT −]] [min, [max, max]] [max, [+,+]] [−, [+,NPT]] [+, [+,+]] [+, [NPT, min]] [min, [TIS,÷]] [÷, [−,+]] [−, [+, max]] [max, [×, ÷]] [÷, [WIQ, +]] [+, [+, PT]] [+, [+, max]] [+, [×, PT]] [max, [W, min]] [min, [×, NPT]] [×, [NPT, NINQ]])

Figure 13: The adjacency lists of the output TGP and LGP heuristics from a run in $\langle Fmean, 0.95\rangle$. The dark shadow highlights the shared adjacency of primitives between the two adjacency lists.

carefully adjusting the proportion of TGP and LGP population for a certain problem, uniformly allocating the training resources to different representations is a relatively good and robust setting for CRGP-TL.

5.6 Example Analyses on Adjacency Lists

The proposed adjacency list-based crossover shares the search information between tree-based and linear representations by the adjacency of primitives. To have a better understanding of knowledge sharing via adjacency lists, this section analyzes the shared knowledge (i.e., primitive adjacency) in two example adjacency lists, where each item in the lists contains two pairs of primitive connections. Fig. 13 shows two adjacency lists of the best-of-run individuals from the two representations, respectively, of the same run for solving the $\langle Fmean, 0.95\rangle$ DJSS problem. If a primitive connection can be seen in both adjacency lists, we highlight the connection with a dark shadow. For example, as the first shadowed item in $\mathbf{L}_{TGP}$ shows the adjacency from "×" to "max", the adjacency items with the same connection in $\mathbf{L}_{LGP}$ are shadowed (e.g., the last item at the second line of $\mathbf{L}_{LGP}$). We can see that the adjacency lists of the output heuristics with tree-based and linear representations have a large number of shared members. For example, both of them prefer concatenating "PT", "NPT", and "NINQ" with "×" and "+", which further form the shared building blocks such as "$[+, [\times, PT]]$" and "$[\times, [NPT, NINQ]]$".

Furthermore, the adjacency lists from different representations have distinct characteristics. Because of short and wide tree structures, the adjacency list of the tree-based representation considers more distinct input features, such as "WKR" and "rFDD". In contrast, the adjacency list of the linear representation uses a large number of "max" and "min" to assemble the final result.

Overall, by exchanging adjacency lists, tree-based and linear representations can 1) learn the shared adjacency of effective solutions and 2) learn the distinct characteristics of the other representation.

6 Conclusions

The main goal of this paper is to verify the effectiveness of a new idea, utilizing the interplay of different GP representations to automatically identify the most suitable representation for the problem at hand. We developed a cross-representation GP method based on tree-based and linear GP representations, denoted as CRGP-TL. Furthermore,

Evolutionary Computation Just Accepted MS.
https://doi.org/10.1162/evco_a_00025
© 2025 by the Massachusetts Institute of Technology

we proposed a novel cross-representation adjacency list-based crossover operator to exchange building blocks between tree-based and linear GP representations in CRGP-TL. To the best of our knowledge, this paper is the first work highlighting that the interplay among different GP representations is useful for improving GP performance.

The experimental studies on symbolic regression and automatic decision rule design show that the proposed CRGP-TL significantly improves the performance of baseline GP methods in the two domains and has a very competitive performance with advanced methods in solving DJSS problems. Further analyses confirm that the cross-representation knowledge sharing helps GP methods have more diverse and effective elite individuals and jump out of local optima.

Our experiments also reveal that the effectiveness of CRGP might be limited by the less effective representations since CRGP-TL has to waste computation resources in those representations. Therefore, we plan to develop more effective collaboration methods among GP representations in the future. Specifically, adaptively and selectively evolving GP representations is a promising research direction to further improve the performance of cross-representation GP methods. We will also extend CRGP to diverse GP representations such as gene expression programming [14], multi-expression programming [34], Cartesian GP [29], and graph-based genetic programming [2].

References

[1] Al-Helali, B., Chen, Q., Xue, B., and Zhang, M. (2021). Multitree Genetic Programming With New Operators For Transfer Learning In Symbolic Regression With Incomplete Data. *IEEE Transactions on Evolutionary Computation*, 25:1049–1063.

[2] Atkinson, T., Plump, D., and Stepney, S. (2018). Evolving Graphs by Graph Programming. In *Proceedings of European Conference on Genetic Programming*, pages 35–51.

[3] Banzhaf, W., Machado, P., and Zhang, M., editors (2024). *Handbook of Evolutionary Machine Learning*. Genetic and Evolutionary Computation. Springer Nature, 1 edition.

[4] Bi, Y., Xue, B., and Zhang, M. (2022). Genetic Programming-Based Evolutionary Deep Learning for Data-Efficient Image Classification. *IEEE Transactions on Evolutionary Computation*. doi:10.1109/TEVC.2022.3214503.

[5] Brameier, M. and Banzhaf, W. (2007). *Linear Genetic Programming*. Springer US.

[6] Cai, X., Gao, L., and Li, X. (2020). Efficient Generalized Surrogate-Assisted Evolutionary Algorithm for High-Dimensional Expensive Problems. *IEEE Transactions on Evolutionary Computation*, 24(2):365–379.

[7] Cazzaro, D. and Pisinger, D. (2022). Variable Neighborhood Search for Large Offshore Wind Farm Layout Optimization. *Computers and Operations Research*, 138:105588.

[8] Chen, Q., Xue, B., and Zhang, M. (2019). Instance Based Transfer Learning for Genetic Programming for Symbolic Regression. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 3006–3013.

[9] Da, B., Gupta, A., Ong, Y. S., and Feng, L. (2016). Evolutionary Multitasking Across Single and Multi-objective Formulations for Improved Problem Solving. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 1695–1701.

[10] D'Ariano, A., Pacciarelli, D., Pistelli, M., and Pranzo, M. (2015). Real-time Scheduling of Aircraft Arrivals and Departures in A Terminal Maneuvering Area. *Networks*, 65(3):212–227.

[11] Fan, H., Xiong, H., and Goh, M. (2021). Genetic Programming-based Hyper-heuristic Approach for Solving Dynamic Job Shop Scheduling Problem with Extended Technical Precedence Constraints. *Computers & Operations Research*, 134:105401.

[12] Feng, L., Huang, Y., Zhou, L., Zhong, J., Gupta, A., Tang, K., and Tan, K. C. (2021). Explicit Evolutionary Multitasking for Combinatorial Optimization: A Case Study on Capacitated Vehicle Routing Problem. *IEEE Transactions on Cybernetics*, 51(6):3143–3156.

[13] Feng, L., Zhou, L., Zhong, J., Gupta, A., Ong, Y. S., Tan, K. C., and Qin, A. K. (2019). Evolutionary Multitasking via Explicit Autoencoding. *IEEE Transactions on Cybernetics*, 49(9):3457–3470.

[14] Ferreira, C. (2001). Gene Expression Programming: a New Adaptive Algorithm for Solving Problems. *Complex Systems*, 13:87–129.

[15] Forstenlechner, S., Fagan, D., Nicolau, M., and O'Neill, M. (2017). A Grammar Design Pattern for Arbitrary Program Synthesis Problems in Genetic Programming. In *Proceedings of European Conference on Genetic Programming*, volume 10196, pages 262–277.

[16] Gupta, A., Ong, Y. S., and Feng, L. (2016). Multifactorial Evolution: Toward Evolutionary Multitasking. *IEEE Transactions on Evolutionary Computation*, 20:343–357.

[17] Hildebrandt, T., Heger, J., and Scholz-reiter, B. (2010). Towards Improved Dispatching Rules for Complex Shop Floor Scenarios - a Genetic Programming Approach. In *Proceedings of the Annual Conference on Genetic and Evolutionary Eomputation*, pages 257–264.

[18] Huang, Z., Mei, Y., Zhang, F., and Zhang, M. (2022a). A Further Investigation to Improve Linear Genetic Programming in Dynamic Job Shop Scheduling. In *Proceedings of 2022 IEEE Symposium Series on Computational Intelligence*, pages 496–503.

[19] Huang, Z., Mei, Y., Zhang, F., and Zhang, M. (2023a). Grammar-guided Linear Genetic Programming for Dynamic Job Shop Scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1137–1145.

[20] Huang, Z., Mei, Y., Zhang, F., and Zhang, M. (2023b). Multitask Linear Genetic Programming with Shared Individuals and its Application to Dynamic Job Shop Scheduling. *IEEE Transactions on Evolutionary Computation*, pages 1–15. doi:10.1109/TEVC.2023.3263871.

[21] Huang, Z., Mei, Y., and Zhang, M. (2021). Investigation of Linear Genetic Programming for Dynamic Job Shop Scheduling. In *Proceedings of IEEE Symposium Series on Computational Intelligence*, pages 1–8.

[22] Huang, Z., Mei, Y., and Zhong, J. (2022b). Semantic Linear Genetic Programming for Symbolic Regression. *IEEE Transactions on Cybernetics*, pages 1–14. doi:10.1109/TCYB.2022.3181461.

[23] Huang, Z., Zhang, F., Mei, Y., and Zhang, M. (2022c). An Investigation of Multitask Linear Genetic Programming for Dynamic Job Shop Scheduling. In *Proceedings of European Conference on Genetic Programming*, pages 162–178.

[24] Jiao, R., Xue, B., and Zhang, M. (2022). A Multiform Optimization Framework for Constrained Multiobjective Optimization. *IEEE Transactions on Cybernetics*, pages 1–13. doi:10.1109/TCYB.2022.3178132.

[25] Kantschik, W. and Banzhaf, W. (2001). Linear-tree GP and Its Comparison with Other GP Structures. In *Proceedings of European Conference on Genetic Programming*, volume 2038, pages 302–312.

[26] Koza, J. R. (1992). *Genetic Programming : On the Programming of Computers By Means of Natural Selection*. Cambridge, MA, USA: MIT Press.

[27] Magalhães, D., Lima, R. H., and Pozo, A. (2023). Creating Deep Neural Networks for Text Classification Tasks Using Grammar Genetic Programming. *Applied Soft Computing*, 135:110009.

[28] Makke, N. and Chawla, S. (2024). Interpretable scientific discovery with symbolic regression: A review. *Artificial Intelligence Review*, 57(1):2.

[29] Miller, J. F. (1999). An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2:1135–1142.

[30] Mladenović, N. and Hansen, P. (1997). Variable Neighborhood Search. *Computers & Operations Research*, 24:1097–1100.

[31] Nguyen, S., Mei, Y., and Zhang, M. (2017). Genetic Programming for Production Scheduling: A Survey with A Unified Framework. *Complex & Intelligent Systems*, 3(1):41–66.

[32] Nordin, P. (1994). A Compiling Genetic Programming System That Directly Manipulates the Machine Code. *Advances in Genetic Programming*, 1:311–331.

[33] Nordin, P. (1997). *Evolutionary Program Induction of Binary Machine Code and Its Applications*. PhD thesis, University of Dortmund.

[34] Oltean, M. and Dumitrescu, D. (2002). Multi Expression Programming. Technical report, Babeş-Bolyai University.

[35] Ong, Y. S. (2015). Towards Evolutionary Multitasking: A New Paradigm in Evolutionary Computation. In *Proceedings of Computational Intelligence, Cyber Security and Computational Models*, pages 25–26.

[36] Pei, W., Xue, B., Zhang, M., Shang, L., Yao, X., and Zhang, Q. (2023). A Survey on Unbalanced Classification: How Can Evolutionary Computation Help? *IEEE Transactions on Evolutionary Computation*, pages 1–21.

[37] Pitzer, E. and Affenzeller, M. (2012). A Comprehensive Survey on Fitness Landscape Analysis. In Fodor, J., Klempous, R., and Suárez Araujo, C., editors, *Recent Advances in Intelligent Engineering Systems*, volume 378, pages 161–191. Springer, Berlin, Heidelberg.

[38] Schauer, J. and Schwarz, C. (2013). Job-shop Scheduling in A Body Shop. *Journal of Scheduling*, 16(2):215–229.

[39] Song, X., Sun, H., Wang, X., and Yan, J. (2019). A Survey of Automatic Generation of Source Code Comments: Algorithms and Techniques. *IEEE Access*, 7:111411–111428.

[40] Sotto, L. F. D. P., Kaufmann, P., Atkinson, T., Kalkreuth, R., and Basgalupp, M. P. (2021). Graph Representations in Genetic Programming. *Genetic Programming and Evolvable Machines*, 22:607–636.

[41] Wei, T., Wang, S., Zhong, J., Liu, D., and Zhang, J. (2021). A Review on Evolutionary Multi-Task Optimization: Trends and Challenges. *IEEE Transactions on Evolutionary Computation*, 26:941–960.

[42] Wilson, G. and Banzhaf, W. (2008). A Comparison of Cartesian Genetic Programming and Linear Genetic Programming. In *Proceedings of European Conference on Genetic Programming*, pages 182–193.

[43] Wittenberg, D. and Rothlauf, F. (2023). Small Solutions for Real-World Symbolic Regression Using Denoising Autoencoder Genetic Programming. In *Genetic Programming. EuroGP 2023*, pages 101–116.

[44] Wu, Y., Ding, H., Gong, M., Qin, A. K., Ma, W., Miao, Q., and Tan, K. C. (2022). Evolutionary Multiform Optimization with Two-stage Bidirectional Knowledge Transfer Strategy for Point Cloud Registration. *IEEE Transactions on Evolutionary Computation*, pages 1–15. doi:10.1109/TEVC.2022.3215743.

[45] Yi, J., Bai, J., He, H., Zhou, W., and Yao, L. (2020). A Multifactorial Evolutionary Algorithm for Multitasking under Interval Uncertainties. *IEEE Transactions on Evolutionary Computation*, 24(5):908–922.

[46] Zhang, F., Mei, Y., Nguyen, S., Tan, K. C., and Zhang, M. (2022a). Multitask Genetic Programming-Based Generative Hyperheuristics: A Case Study in Dynamic Scheduling. *IEEE Transactions on Cybernetics*, 52:10515–10528.

[47] Zhang, F., Mei, Y., Nguyen, S., and Zhang, M. (2022b). Multitask Multiobjective Genetic Programming for Automated Scheduling Heuristic Learning in Dynamic Flexible Job-Shop Scheduling. *IEEE Transactions on Cybernetics*, pages 1–14. doi:10.1109/TEVC.2023.3263871.

[48] Zhang, F., Mei, Y., Nguyen, S., and Zhang, M. (2023). Survey on Genetic Programming and Machine Learning Techniques for Heuristic Design in Job Shop Scheduling. *IEEE Transactions on Evolutionary Computation*, 28(1):147–167.

[49] Zhang, F., Nguyen, S., Mei, Y., and Zhang, M. (2021). *Genetic Programming for Production Scheduling*. Springer Singapore.

[50] Zhong, J., Feng, L., Cai, W., and Ong, Y. S. (2020). Multifactorial Genetic Programming for Symbolic Regression Problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(11):4492–4505.