A Data Structure for Improved GP Analysis via Efficient Computation and Visualisation of Population Measures

Anikó Ekárt[1] and Steven Gustafson[2]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences,
1518 Budapest, P.O.B. 63, Hungary
ekart@sztaki.hu
[2] School of Computer Science & IT, University of Nottingham,
Jubilee Campus, Wollaton Rd, Nottingham, NG81BB, United Kingdom
smg@cs.nott.ac.uk

Abstract. Population measures for genetic programs are defined and analysed in an attempt to better understand the behaviour of genetic programming. Some measures are simple, but do not provide sufficient insight. The more meaningful ones are complex and take extra computation time. Here we present a unified view on the computation of population measures through an information hyper-tree (iTree). The iTree allows for a unified and efficient calculation of population measures via a basic tree traversal.

1 Introduction

"Things should be as simple as possible, but not simpler."
Albert Einstein

A population search method using variable length representation (such as genetic programming) requires expensive measures to collect, mine and visualise dynamics due to the repeated traversal of individuals making up the population. For the researcher, it would be useful to quickly have an overview of population dynamics, which often involves complex computations. However, it is essential to employ measures which are easy to use, intuitive, and efficient to compute in order to reduce the complexity and expense of such analysis. Implementing such measures is usually difficult and time consuming. Instead, the complexity of the analysis, algorithm or problem are typically reduced to allow for detailed analysis.

We introduce an advanced data structure that is efficient to maintain, offers a compact view on a population of tree structured genetic programs and allows for the efficient computation of many population measures. In this way, exploratory analysis beyond simple measures (like fitness, node counts or diversity based on uniqueness) becomes more accessible to the researcher.

Our data structure captures the population information needed to compute several simple and complex measures. The data structure (the information tree, or "iTree") is realised as a "hyper-tree" on the population of trees. The iTree represents the information obtained from a complete traversal of the individuals in the population. The uses of the iTree are two-fold:

- the iTree makes the computation of many useful population measures possible and efficient and
- the iTree can be visualised, offering compact views of the population.

We first introduce the information hyper-tree and demonstrate how it improves the efficiency of several measures. We then discuss several more complex and expensive measures, which become accessible and intuitive with the help of the iTree. We also describe typical situations that call for iTree visualisation.

2 The Information Hyper-tree of a Population

We introduce the information hyper-tree (iTree) for a population as a data structure that collects important details of the individuals making up the population in one easily accessible place. We construct the iTree of a population of genetic programs based on two principles:

1. *The structure of the iTree must be such that it incorporate the structure of any tree in the population.* Therefore, the iTree will have a node at some location if there is at least one member of the population having a node at the same location.
2. *Each node of the iTree should capture the population information related to that particular node position.* In the basic case, this information is the number of trees in the population that have a node at the given location.

In Fig. 1 we show example iTrees for three small populations of genetic trees (for a symbolic regression problem). For simplicity, we only look at binary trees, i.e. problems where all functions accept two arguments. The measures presented in Fig. 1 will be explained later.

The iTree can be constructed for any set of genetic trees, not only for the whole population. For example, the iTree corresponding to the most fit individuals during a run could provide information about the expected structure of the solution to the problem. The iTree corresponding to all individuals visited during a run could provide information about the region of the tree search space covered by the run. The iTree corresponding to the solutions of a number of runs could uncover structural similarities or differences of the found solutions and contribute to the understanding of GP-hardness, for example by complementing Daida et al.'s study [4].

We construct the iTree in a top-down manner, starting from the root. Consequently, the iTree will reflect the top-down structure of the represented trees. This approach is the opposite of Keijzer's bottom-up method [6]. He builds the trees starting from the leaves, and by representing only once the common subtrees that occur in different individuals he can space-efficiently represent a population. In the mean time, the iTree's purpose is to make possible the computation of complex measures for any set of genetic trees. As subtree-based measures are not that straight-forward to compute with the iTree, the two methods could be used in conjunction.

In the subsequent sections we shall describe the computation of population measures based on the iTrees. We shall refer to the iTrees of populations but the measures for iTrees associated to other sets of trees can be computed similarly.

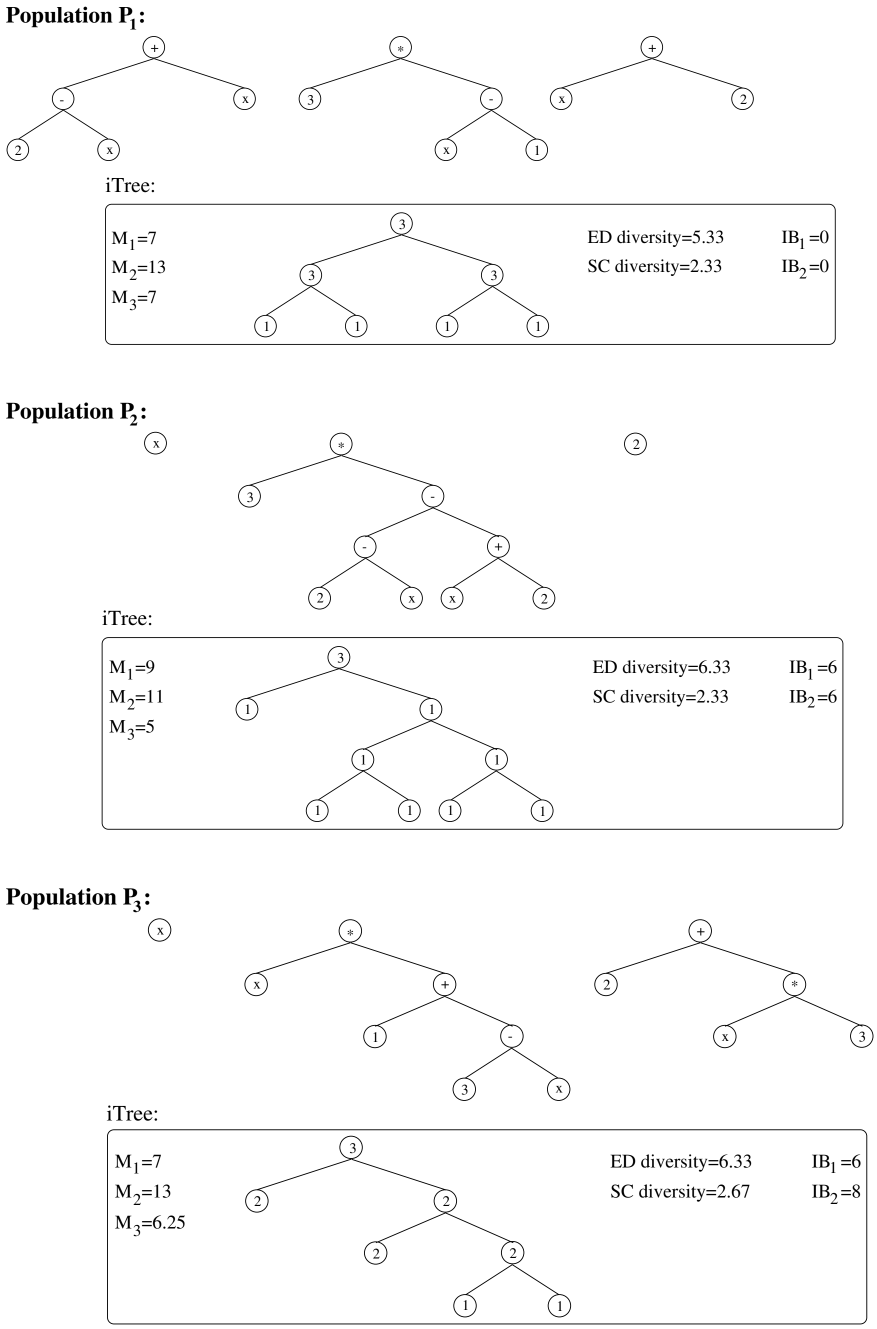


Fig. 1. The iTrees and measures corresponding to three populations of three genetic trees each

3 The Node Coverage of a Population

There are a number of meaningful measures that can be easily computed from the basic information stored in the nodes of the iTree. These measures can be used in analysing how well a population explores the search space of tree structures. We are able to find answers to the following questions related to a population P with corresponding $iTree$:

- How many node positions are being explored in the tree search space? This is the number of node positions in a binary tree that the population samples by at least one member. This measure can be found by simply counting the nodes of the $iTree$,

$$M_1(P) = \sum_{A \in iTree} 1.$$

- How many genetic nodes are there in the population? We only have to sum up the values stored in the nodes of the $iTree$:

$$M_2(P) = \sum_{A \in iTree} n_A.$$

  When comparing two populations of trees, we can say which one is larger in terms of node numbers. Note that in order to obtain the total number of nodes in a population without using the iTree, one would have to sum up the sizes of individuals in the population.
- How full is the $iTree$? We can compute the degree of fullness as:

$$M_3(P) = \frac{1}{size(P)} \sum_{A \in iTree} \frac{1}{2^{depth(A)}} n_A.$$

  In order to obtain this measure without using the iTree, one would have to do a traversal of all trees in a population. For a full $iTree$, $M_3(P) = depth(iTree)$. For very sparse trees, $M_3(P) \to 1$.

If two populations P_1, P_2 have values $M_1(P_1) < M_1(P_2)$, $M_2(P_1) > M_2(P_2)$ (as in Fig. 1), then the second population explores a larger region of the tree search space, but the first population performs a better exploration through more representatives of the nodes. If $M_1(P_1) = M_1(P_3)$, $M_2(P_1) = M_2(P_3)$ and $M_3(P_1) > M_3(P_3)$ (see Fig. 1) then the first population has more nodes at lower depth, i.e. has more shallower and fuller trees than the second population. Population P_3 may contain deeper and sparser trees than population P_1.

3.1 Entropy

Suppose the iTree also contains information about the distribution of terminal and function values that are covered by each node in the population. We could associate an entropy value to each iTree node that can show how biased the population is toward specific values in the nodes. For a given node A, the distribution of values over the set of functions and terminals $F \cup T$ is given by $D : F \cup T \rightarrow \mathbb{N}$, where $D(s)$ is the number of genetic trees in the population containing symbol s in the location corresponding to node A. The entropy of node A is then

$$E(A) = - \sum_{s \in F \cup T} \frac{D(s)}{\sum_{v \in F \cup T} D(v)} log \frac{D(s)}{\sum_{v \in F \cup T} D(v)}$$

where log is the logarithm with the base the total number of functions and terminals $|F \cup T|$. The closer the entropy value is to 1, the more uniform the distribution is and the closer the entropy value is to zero, the more biased the distribution is. The entropy is also a measure of node content diversity: for a given node it has a larger value for a better coverage of the function and terminal sets and a lower value for a worse coverage, respectively.

4 Structural Diversity of a Population

Measuring genotype diversity in tree-based genetic programming can be very time-consuming. Structural diversity is usually measured based on pairwise distances between individuals in a population[1, 5, 8]. Usually, the average distance of two individuals in a population is employed. Therefore, computing diversity is very expensive: if the population size is N, there are $N \times (N - 1)$ pairs of individuals. For each pair T_1, T_2, the time complexity of obtaining their edit distance is $O(|T_1| \times |T_2|)$. Similarly to Wineberg and Oppacher's results on genetic algorithms [10], we can significantly reduce the computation time of structural diversity when using the iTree.

4.1 Edit Distance

The edit distance of two labeled trees is defined as the cost of shortest sequence of editing operations that transform one tree into the other [7, 9]. The editing operations are deleting a node, inserting a node, or changing the label of a node. In the basic case each operation is considered with the same cost. In order to compute the structural diversity of a population, we need to maintain the symbol (terminal and function) distribution in each node of the iTree. Normally, diversity would be calculated as the average edit distance over all pairs of individuals in the population. When comparing two individual trees node by node, three cases can occur:

1. Both trees have the same symbol at the examined position, the node has no contribution to the edit distance;
2. The trees have different symbols at the examined position, so the cost of changing a label is added to the edit distance; or
3. One tree has no node at the given position, so the cost of adding/deleting a node is added to the edit distance.

At the node level in the iTree, each individual is represented by the number of occurrences of the individual's symbol in that position. The same edit distance diversity can be calculated by traversing the iTree and summing up the nodes' contributions. Given the distribution of symbols for a node, obtaining the node's contribution can be formulated as counting the number of pairs of non-identical symbols encountered in that position in the iTree.[3] The number of pairs in a set of N symbols is $N\,(N-1)/2$. Similarly, the number of pairs in a subset of $D(s)$ identical symbols is $D(s)\,(D(s)-1)/2$. So, the number of non-identical pairs is

$$
\begin{aligned}
&N\,(N-1)/2 - \textstyle\sum_{s\in F\cup T} D(s)\,(D(s)-1)/2 \\
&\quad = (1/2)\,\left(N^2 - N - \sum D(s)^2 + \sum D(s)\right) = (1/2)\,\left(N \sum D(s) - \sum D(s)^2\right) \\
&\qquad\qquad\qquad = (1/2)\,\sum \left(N\,D(s) - D(s)^2\right) = (1/2)\,\sum D(s)\,(N - D(s)).
\end{aligned}
$$

The algorithm for computing diversity as the average edit distance between two individual trees of a population of N trees with corresponding $iTree$ is presented below. We denote by $iNodes$ the number of individuals in the population, which sample the root node of the $iTree$. The time complexity of the algorithm is $O(|F \cup T| \times size(iTree))$.

```
Diversity(iTree, N)
begin
 dist=0;
 if iNodes < N
   dist := dist + iNodes x (N - iNodes);
 for each symbol s encountered in the root of iTree D(s) times
   dist := dist + D(s) x (N - D(s));
 dist := dist / (N x (N - 1));
 if the root has nonempty left child iLeft
   dist := dist + Diversity(iLeft, N);
 if the root has nonempty right child iRight
   dist := dist + Diversity(iRight, N);
 return dist;
end
```

4.2 Distance Based on Structural Comparison

Instead of the edit distance we can use the distance based on structural comparison described in[5]. The distance of two binary trees A and B, with the two subtrees denoted as $.Left$ and $.Right$ is defined as:

$$
dist(A,B) = \begin{cases} d(root(A), root(B)) & \text{if both } A \text{ and } B \text{ are leaves} \\ d(root(A), root(B)) + & \\ \frac{1}{K} \times (dist(A.Left, B.Left) + dist(A.Right, B.Right)) & \text{otherwise.} \end{cases}
$$

In the simple case, when we just count the differences, i.e. $d(x,y) = \delta_{xy}$ (Kronecker delta), if we use the iTree for computing the average distance of two trees in a population, the contribution of a node is the same as in the case of the edit distance. The only difference in the algorithm is that the diversity of subtrees is discounted:

[3] For unified treatment, we can assume that each node in the iTree represents $iNodes$ nodes, $N - iNodes$ being the number of "empty" symbols, or number of individuals in the population which have no node at that location.

```
 if the root has nonempty left child iLeft
   dist := dist + 1/K x Diversity(iLeft, N);
 if the root has nonempty right child iRight
   dist := dist + 1/K x Diversity(iRight, N);
```

In the more general case, computing the contribution of a node becomes more complex,the time needed for this operation becomes $O(|F \cup T|^2)$ and consequently the time complexity of the algorithm becomes $O(|F \cup T|^2 \times size(iTree))$.

5 Distance of Two Populations

As emphasised by Wineberg and Oppacher [10], the distance of two – consecutive – populations can play an important role in understanding the dynamics of evolutionary computation methods. Instead of computing pairwise distances between individuals from the two populations, we can easily obtain the distance of the two populations by traversing their iTrees in parallel.

Normally, the distance between two populations is computed as the average distance between any two individuals from the two populations. Computing the average distance diversity of a population is a special case of computing the distance of two populations, namely, when the two populations are identical and when the distance of an individual to itself is not counted. If we use the iTrees of the two populations, we obtain the following algorithm for computing the edit distance of two populations by generalising the algorithm presented in Section 4.1.[4] The time complexity of the algorithm is $O(|F \cup T| \times (size(iTree1) + size(iTree2)))$.

```
Pop_Dist(iTree1, iTree2, N1, N2)
begin
 dist=0;
 if iNodes1 < N1
   dist := dist + (N1 - iNodes1) x iNodes2
               + iNodes1 x (N2 -iNodes2);
 for each symbol s, D1(s) times in iTree1, D2(s) times in iTree2
   dist := dist + D1(s) x (N2 - D2(s))
               + (N1 -D1(s)) x D2(s);
 dist := dist / (2 x (N1 x N2));
 if at least one root has nonempty left child (iLeft1 or iLeft2)
   dist := dist + Pop_dist(iLeft1, iLeft2, N1, N2);
 if at least one root has nonempty right child (iRight1 or iRight2)
   dist := dist + Pop_dist(iRight1, iRight2, N1, N2);
 return dist;
end
```

In order to obtain the distance based on structural comparison, we only have to modify the recursive calls to include the discounts, similarly to the structural diversity presented in Section 4.2.

6 The Imbalance of a Population

The iTree of a population can provide information about how the population samples different tree structures. A very unbalanced iTree suggests a biased sampling (specific,

[4] Whenever one subtree is empty, we make the recursive call for the null subtree representing zero nodes.

mostly sparse tree structures are often present in the population), whereas a balanced iTree suggests uniform sampling of nodes. The imbalance of a population can be seen as an indicator of the structural diversity of the population: (1) A largely unbalanced iTree suggests the presence of many similar trees in a population (low diversity). (2) A balanced iTree corresponds to a population that covers the full tree structure. In an extreme, but very unlikely case this could be the result of the population containing only full trees. Otherwise, the population contains all sorts of structures, which cover together the full tree structure (high diversity).

6.1 Original Imbalance

Colless [2] defines an imbalance measure for a phylogenetic tree as the sum of absolute differences between the sizes of the two subtrees located at each bifurcation in the tree under consideration. We use the same imbalance measure for our data structure and denote it by IB_1. We consider the size of a subtree as the number of nodes contained in it. A fully balanced iTree has the same number of nodes in all subtrees that have the same parent. In the case of binary trees, the only fully balanced tree of some given depth is the complete tree of the same depth. A population of trees corresponding to a fully balanced iTree explores all the possible nodes in the program tree structure to some extent. The more unbalanced an iTree is, the more unexplored regions exist in the "structure space" of program trees. In order to say something about the extent to which the program tree structure is being explored, or more exactly, how uniformly the program tree structure is being explored, we need a more detailed measure of imbalance.

6.2 A More Specific Imbalance

We consider the same imbalance measure for a population's iTree, with the exception that instead of the size of a subtree, we use the total number of genetic tree nodes contained in that subtree (that is, the sum of node numbers for each iTree node). We denote this type of imbalance as IB_2 and show three examples in Fig. 2. The most unbalanced iTree would have all nodes concentrated in one subtree. But if a node of the iTree has any children, it must have two children, both representing the same number of genetic nodes, due to our restriction of binary functions. If we denote by n_i the number

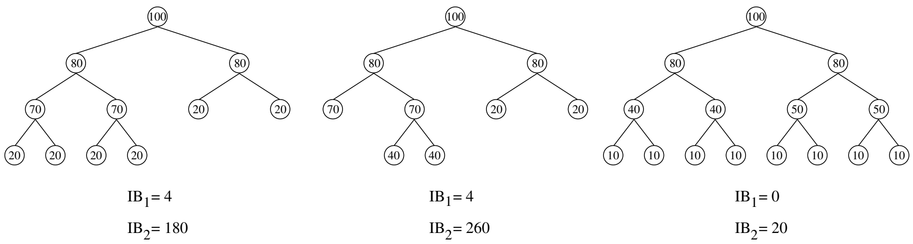
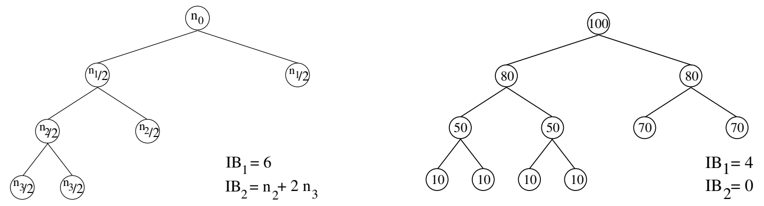


Fig. 2. Examples of different iTrees corresponding to the same total number of nodes

of genetic tree nodes at depth i in an iTree, the largest possible imbalance IB_2 for a tree of depth k is $\sum_{i=2}^{k}(i-1)\times n_i$ (see the example for depth 3 in Fig. 3(a)). [5]

A completely balanced iTree is a full tree with uniformly distributed genetic nodes at each depth, with corresponding $IB_2 = 0$. However, there exist unbalanced iTrees with $IB_2 = 0$, such as the example shown in Fig. 3(b). In such cases, the original imbalance measure $IB_1 > 0$. The two measures are insufficient by themselves, but, taken together, they can show whether a population is exploring all regions of the structure search space (IB_1), and to what extent the actually represented regions are being explored (IB_2). If an iTree has $IB_1 = 0$ and $IB_2 = 0$, the iTree is a full tree of some depth d and all node locations at some depth $i \leq d$ are explored to the same extent in the population. This does not necessarily involve just full trees in the population (see Fig. 1 for example).



(a) The most unbalanced iTree of depth 3

(b) An unbalanced iTree with $IB_2 = 0$

Fig. 3. Examples of unbalanced iTrees

7 Visualisation of the Information Hyper-tree

In order to better analyse and understand the dynamics of genetic programming runs, we could graphically show the iTrees or their parts associated to certain groups of genetic trees. In many cases it is important to see the actual tree structures. For instance, the following questions highlight the need for good, and sometimes complex, visualisation of the population:

1. *What are the most common structures in the best genetic programs that were encountered during a run?* In order to get an answer, we have to first quantify "most common" and "best", such as the nodes that occur in at least 80% of the trees whose fitness is below 0.1 (if we are minimising fitness). Then we construct the iTree corresponding to the genetic trees of desired fitness encountered during the run and visualise the nodes A of the iTree with $\frac{n_A}{n_0} > 0.8$. A small tree would show that the trees are similar only at very low depth, meaning that a diverse set of solutions has been found. A large, perhaps sparse tree would show that genetic programming conducted a local search around one optimum, where the found structure is a good structure. In Fig. 4 we show the most common best tree structure that emerged in

[5] Proof. The difference in node numbers at depth i of a fully unbalanced iTree is $\sum_{j=i+1}^{k} n_j$. We complete the proof by summing up these differences.

a run of a symbolic regression problem. We qualified all trees with fitness below 0.05 in the best category. Out of the 5100 trees evaluated during the run, 2163 trees were considered best. The large common part suggests a local search around a good structure.

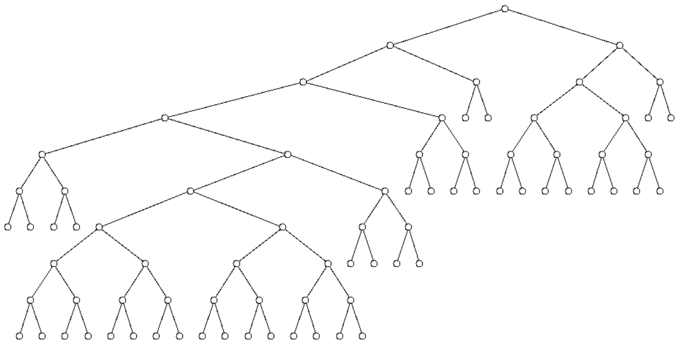


Fig. 4. Example good common structure evolved in a GP run

2. *What makes a good program different from a bad program?* One possibility is to look at the most common part of the iTree corresponding to the best trees and the most common part of the iTree corresponding to the worst trees encountered during the run. One could also do a parallel traversal of the two iTrees and show only the nodes that are common in one group, but not in the other. For the example discussed at (1), 315 trees were very unfit, their most common structure was the full tree of depth 3. The set of unfit trees was quite diverse, so we can only say that the structure shown in Fig. 4 is good, and the structures different from this one might be bad.
3. *How early in the run do the good structures emerge?* Answering this question involves a comparison between the iTree of (1) and the iTrees of populations in subsequent generations.
4. *If there are common structures during the run, do they heavily depend on the initial population?* As in the case of question (2), we first have to look at the iTree of the run (or more exactly its most common parts), and then compare it with the iTrees of populations in subsequent generations. Thus, we could find when these structures first appeared. The earlier these structures emerge, the more likely it is that they depend on the initial population.

8 Case Study

Here we present an analysis based on iTrees for a symbolic regression problem. We use the polynomial $(x + 0.2)^2 (x - 0.5) (x + 0.5) (x - 0.7)$ with $x \in [-1, 1]$. We show the results for 50 independent runs of simple genetic programming. The initialisation method is ramped half and half, populations have size 100, runs are allowed 50 generations. Crossover probability of 90% and mutation probability of 10% are employed. We analyse the iTrees for the runs and the iTrees corresponding to the best and the worst genetic programs encountered during the run, respectively. We classified as best the solution-quality trees (with $fitness \leq 0.03$) and as worst the useless trees (with $fitness > 1$). 37 runs were successful, 4 runs had acceptable solution and 9 runs were unsuccessful. In the successful runs, the good trees significantly outnumber the useless trees.

In Table 1 we show the values for five iTree-based measures described in Sections 3 and 6. For all measures, the difference between the value for the best iTree and the worst iTree is significant. The best iTree has more nodes than the worst iTree, the total number of nodes in the best trees is larger than the total number of nodes in the worst trees, meaning that during evolution more time is spent on the good trees than on the bad ones. The best trees are larger and fuller than the worst ones. As shown in Fig. 5(a), the runs producing better solutions evolve a larger range of mostly fuller trees. Also, the best iTrees are fuller than the iTrees of the runs.

Table 1. The measures computed on iTrees of runs, best trees and worst trees encountered during the runs. The average of 50 runs and corresponding confidence interval (with 95% confidence) is presented in each case

	M_1 [$\times 10^2$]	M_2 [$\times 10^3$]	M_3	IB_1 [$\times 10^3$]	IB_2 [$\times 10^4$]
Run	59.7 ± 11.4	389 ± 73.1	4.24 ± 0.26	35.6 ± 6.8	185.6 ± 42.4
Best	31.8 ± 6.5	249.9 ± 53.9	4.93 ± 0.23	23.5 ± 4.7	143.3 ± 33.6
Worst	10.6 ± 2.6	12.8 ± 4.6	3.78 ± 0.22	6.3 ± 1.6	5.1 ± 2.3

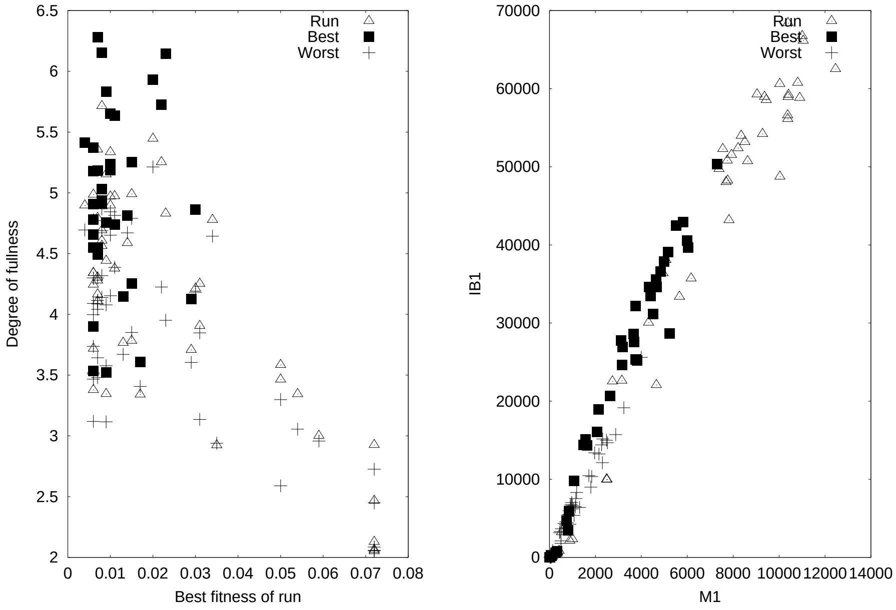
An interesting linear relationship emerges between the imbalance (IB_1) and the number of nodes (M_1) in the iTrees for all three cases (Fig. 5(b)). The more nodes are investigated, the more unbalanced the iTree becomes. This is somewhat expected and is in line with the literature on code growth in genetic programming.

9 Conclusions

In this paper, we motivated the need for an intermediate data structure by pointing to the complexity required to conceive and develop population measures and visualisations. In the absence of efficient and intuitive methods, researchers often reduce the complexity of other aspects to make analysis tractable. However, by using the proposed iTree, several population measures become intuitive and more efficient than the traditional traversals of the population. Future genetic programming systems can make use of a standardised data structure, such as the iTree, to allow quicker development and sharing of methods and measures to improve the dissemination of scientific ideas.

References

1. E. Burke, S. Gustafson, G. Kendall, and N. Krasnogor. Advanced population diversity measures in genetic programming. In J. M. Guervós et al., editors, *7th International Conference on Parallel Problem Solving from Nature*, volume 2439 of *LNCS*, pages 341–350, 2002.
2. D. H. Colless. Review of phylogenetics: The theory and practice of phylogenetic systematics. *Syst. Zool.*, 31:100–104, 1982.



(a) The degree of fullness of the iTrees vs. the quality of solutions

(b) The linear relationship between imbalance and number of nodes

Fig. 5. The resulting iTrees for the runs, best and worst trees. Each triangle represents the iTree of one run, each square the iTree of best trees encountered in one run, and each plus the iTree of worst trees encountered during a run

3. J. Daida, A. Hilss, D. Ward, and S. Long. Visualizing tree structures in genetic programming. In E. Cantú-Paz et al., editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1652–1664, 2003.
4. J. Daida, H. Li, R. Tang, and A. Hilss. What makes a problem GP-hard? validating a hypothesis of structural causes. In E. Cantú-Paz et al., editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1665–1677, 2003.
5. A. Ekárt and S. Németh. Maintaining the diversity of genetic programs. In J. Foster et al., editors, *Proceedings of the 5th European Genetic Programming Conference*, volume 2278 of *LNCS*, pages 162–171, 2002.
6. M. Keijzer. Efficiently representing populations in genetic programming. In P. J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 13, pages 259–278. MIT Press, 1996.
7. S.-Y. Lu. The tree-to-tree distance and its application to cluster analysis. *IEEE Transactions on PAMI*, 1(2):219–224, 1979.
8. U.-M. O'Reilly. Using a distance metric on genetic programs to understand genetic operators. In J. R. Koza, editor, *Late Breaking Papers at the 1997 Genetic Programming Conference*, pages 199–206, 1997.
9. S. M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.
10. M. Wineberg and F. Oppacher. Distance between populations. In E. Cantú-Paz et al., editors, *GECCO-2003*, volume 2724 of *LNCS*, pages 1481–1492, 2003.