Visualisation of Combinatorial Program Space and Related Metrics

A.V. Husselmann and K.A. Hawick
Computer Science, Massey University, North Shore 102-904, Auckland, New Zealand
email: { a.v.husselmann, k.a.hawick }@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

Abstract—*Searching a large knowledge or information space for optimal regions demands sophisticated algorithms, and sometimes unusual hybrids or combined algorithms. Choosing the best algorithm often requires obtaining a good intuitive or visual understanding of its properties and progress through a space. Visualisation in combinatorial optimizers is more challenging than visualising parametric optimizers. Each problem in combinatorial optimisation is qualitative and has a very different objective, whereas parametric optimizers are quantitative and can be visualised almost trivially. We present a method for visualising abstract syntax trees in an interactive manner, as well as some certain enhancements for evolutionary algorithms. We also discuss the use of this in improving the convergence performance of a Geometric Particle Swarm Optimiser.*

Keywords: combinatorial information; knowledge engineering; visualisation, genetic programming, optimization.

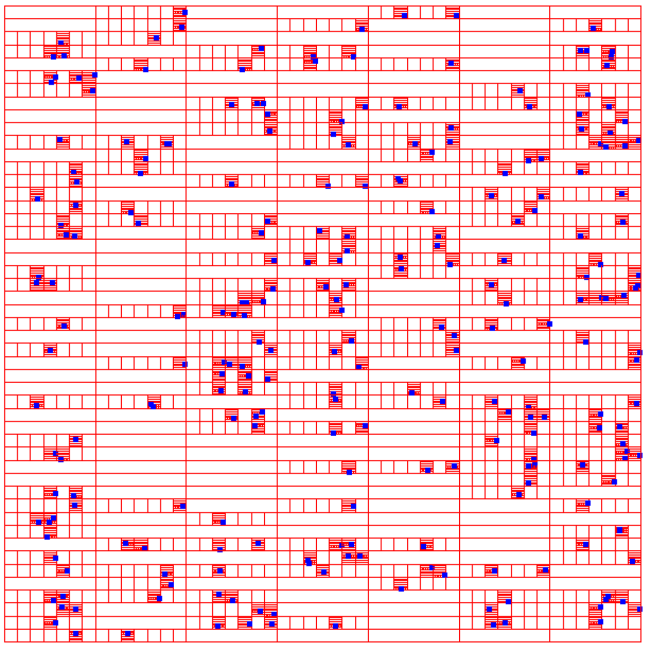
1. Introduction

Combinatorial optimization[1] and the search methods associated with it remain important aspects of information and knowledge engineering. Obtaining a visual representation and hence an understanding of algorithms in combinatorial optimization remains a difficult challenge as the scale and complexity of the problems one wishes to tackle increases. A visual rendering of an algorithm can be an important means of assessing its suitability for a particular problem, particularly if the rendering can be formed in near interactive time and the human user is able to form an impression of an algorithm's progress – or lack of it from a recognizable visual pattern.

Much research has been dedicated towards furthering combinatorial optimizers, with classical problems such as the Traveling Salesman Problem (TSP) [2], [3], the Knapsack problem [4] and the Prisoner's Dilemma [5]. The primary focus of these problems is the search for a global optimum, analogous to that of parametric optimizers such as Kennedy and Eberhart's Particle Swarm Optimiser [6].

John Koza's pioneering work of 1994 [7], perhaps greatly inspired by earlier work of John Holland on the Genetic Algorithm [8] has seen the advent and widespread uptake and use of of Genetic Programming (GP) [9]. GP is a technique used to evolve programs to solve particular problems. Since the introduction of this algorithm, it has been used for solving many problems, such as evolving soccer Soft Bots [10], [11] for competitions, model induction [12], intrusion detection [13], modeling land change in Mexico [14], image enhancement [15] and many more.

Fig. 1
VISUAL REPRESENTATION OF A GENERATION OF AGENTS USING A TREE STRUCTURE.



Many variations and improvements of the GP algorithm have been proposed in the past including Cartesian GP [16], distributed CUDA-based GP [17], a quantum-inspired linear GP [18], Strong GP [19] (a restrictive version of GP), as well as other GPU-based implementations [20], [21].

Visualisation of GP has typically been restricted to visualisation of the fitness function itself. In simulations such as soccer Softbots [10], [11], it is attractive to view the behaviour of the robots themselves, as it gives a good indication of the running efficacy of GP. Based on this information, one can sometimes infer modifications to parameters such as mutation and crossover rates.

In analysis of GP and its related variations, quantitative metrics typically take the place of visualisation. These also give valuable insights into the behaviour of the algorithm.

Techniques such as Landscape Analysis have long been an area of research, and was applied to GP in 1994 by Kinnear [22], the same year that GP was introduced by Koza [7]. In the article by Kinnear, the author also discussed comparing the difficulty of various fitness landscapes by plotting the cumulative probability of success (CPS) for each. Gustafson [23] presented a thorough analysis of diversity metrics in Genetic Programming which include unique programs, ancestral analysis, edit distances, and others.

Our efforts are focused on visualisation of the candidate programs as they are modified by genetic operators. We anticipated that this would assist in verifying the behaviour of each operator, as well as tuning it so as to maximise constructive recombination between candidates.

In Section 2, we describe the algorithms that we apply our visualisation and metrics to. This includes the Geometric Particle Swarm Optimiser using Karva for representation, as well as a Genetic Programming implementation also using Karva. In Section 3 we proceed to discuss our visualisation method, and related metrics, and following this we present and discuss some screen dumps of the visualisation. Finally, in Sections 5 and 6, we discuss our methods, and conclude with some possible future work.

2. Genetic Programming Background

We summarise the genetic programming algorithms of interest to us in this work and give some background and references on their main properties relevant to our test-bed system and visual rendering implementation.

Our test-bed algorithms include a data-parallel implementation of Genetic Programming using *Karva*[24] for representation of programs (which we shall denote K-GP) [25], as well as a data-parallel Geometric Particle Swarm Optimiser also using Karva (which we shall denote using K-GPSO) [26]. We present here a brief overview of these algorithms.

Karva is a program representation language developed by Ferreira in her Gene Expression Programming (GEP) algorithm [24]. GEP is attractive mainly for its representation, which has inherent support for introns in its representation; which brings it closer to the biological analogy of evolution. It is also attractive for its extremely simple and elegant crossover and mutation operators.

Since both K-GP and K-GPSO operate on the space of Karva programs (otherwise known as K-expressions), the main difference between these algorithms is in its recombination phase. K-GP relies on a tournament selection operator, followed by simple one-point crossover and point mutation. The K-GPSO operates using a multi-parent crossover with the global optimum (*gBest*) and a personal optimum (*lBest*), and a current position, analogous to the original PSO. A perturbation in solution space is accomplished using point mutation. The weighted multi-parent crossover operator we use is the one presented by Togelius in his paper introducing Particle Swarm Programming [27] from the concept of a Geometric Evolutionary Algorithm first presented by Moraglio in his thesis of 2007 [28]. The concept of a Geometric optimiser is essentially a method by which a parametric optimiser such as the PSO by Kennedy and Eberhart [6], [29] can be adapted for searching in an arbitrary space.

Part of our interest in developing visualisations and metrics for Evolutionary Algorithms (EAs) is the advent of the very recent concept of Geometric EAs. Poli and colleagues [30] have conceded that it is too early to assert the efficacy of Geometric EAs over traditional related algorithms; which has inspired interest in more metrics and visualisations, as well as new algorithms such as the K-GP and K-GPSO.

Both the K-GP and K-GPSO are implemented on Graphical Processing Units (GPUs) to improve wall-clock performance, but as we are only concerned with convergence performance, we omit a detailed discussion on this, and instead refer the reader to [25], [26], [31] for more detail.

The fitness function we use is a modified Santa Fe Ant Trail in 3D, where terminal symbols are: `Move`, `Right`, `Left`, `Up`, `Down`, and non-terminal symbols (functions) are: `IfFoodAhead` and `ProgN2`. The function `IfFoodAhead` executes its first argument if there is food directly ahead of the agent, and the second argument otherwise. `ProgN2` simply executes its arguments in order. The object of this simulation is to obtain an agent which is as effective as possible for picking up so-called "food" items scattered throughout the space. We omit a more thorough discussion of how this fitness function is implemented.

An example of a *Karva*-expression or k-expression which encodes a certain candidate program is as follows:

```
0123456789
PPIPMMRMML
```

This program is shown as a visual interpretation in Figure 2. It is a highly efficient program for solving this particular problem. The first line of the code above is simply an indexing convenience, whereas the second line is the program itself. The string of symbols is interpreted into a tree (as shown in Figure 2), and then executed in the normal fashion. The tree is constructed level by level, and filling arguments from the k-expression from left to right. This tree is often known as the phenotype for a particular candidate, whereas the string of symbols shown above is the genotype. It is important to note that the symbol at index 9 is not expressed in the phenotype, however, with an appropriate mutation, this symbol can easily be re-introduced into the phenotype.

Point mutation and one-point crossover is almost trivially easy on a representation like this. Point mutation is simply an exchange of one symbol with another uniform-randomly chosen symbol. One-point crossover involves choosing a random crossover site, and exchanging information between

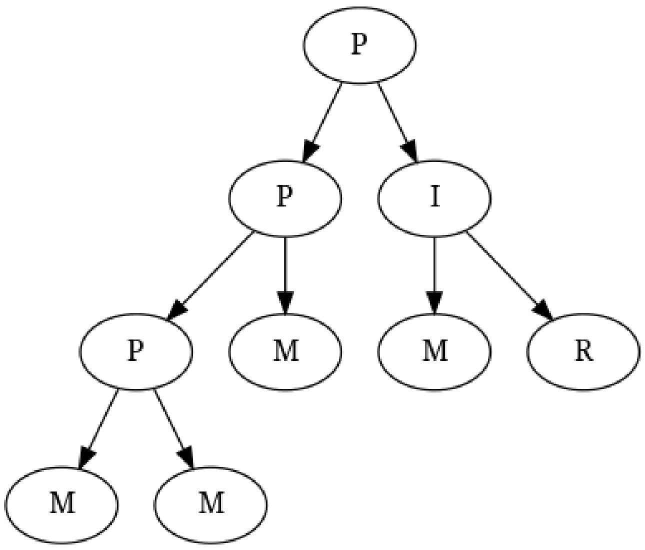


Fig. 2
A HIGHLY EFFECTIVE GENERATED AGENT. THE k-EXPRESSION FOR THIS IS *PPIPMMRMM*.

two candidates about this point.

It is important, however, to maintain a head and tail section in this expression, so that it is guaranteed that all functions in phenotype will have enough arguments supplied to them. Details of this is out of scope here, but it is important to note that, like other Genetic Programming approaches, Karva also has some idiosyncrasies.

3. Visualisation Method

Our method for visualising program space involves a successive subdivision of a 2D grid, where each subdivision represents the selection of a different codon or symbol. We have specifically engineered this method for *karva*-expressions, but it can easily extend to any other abstract syntax tree representation including pointer trees.

Figure 1 shows an example of what a randomly initialised population of candidate programs could look like. In this example, a dot represents a single program. The space is divided in a horizontal fashion, for selecting the first codon, then vertical for the second codon, and so forth, until all codons have been selected, at which point a dot is placed. It is worthwhile to note that in doing this, we are effectively viewing a combinatorial problem as a parametric one, where differences in programs are represented as spatial differences instead.

To further illustrate our method, we present Algorithms 1 and 2. Algorithm 1 shows the process by which we add an expression to the tree-based data-structure of the visualiser. Algorithm 2 is the method by which we actually draw the data-structure to the screen. We keeps Algorithms 1 and 2 separate in the implementation, so that interactive use of the program is more streamlined. The data-structure we use is similar in concept to k-D trees, where space is successively divided along each of the principal axes.

Also, to indicate candidate movement through this pseudo-space in successive generations, we draw a line from the previous candidate to the new candidate in each generation. This makes certain dynamics of EAs more clear, particularly the K-GPSO, which we discuss later.

In summary, for a new expression to be added to the program space visualisation, the space is first divided into n sections vertically, where n is the number of terminal and non-terminal symbols. Each section represents a symbol. The first symbol in the expression determines the section next divided. Suppose this is the third section from the top. This section is then divided into n sections in a horizontal fashion. The next symbol in the expression determines which section will then be divided further, and so forth. Finally, when no symbols remain in the expression, a dot is drawn to indicate the location of the expression.

Algorithm 1 Adding an expression to the data-structure.

```
with $n$ candidate programs
with $p$ as the top-level symbol drawable
set $c$ = $p$
**for** $i = 0$ to $n$ **do**
  with $m$ symbols per program
  exp = programs[i]
  **for** $j = 0$ to $m$ **do**
    nextindex = getCodonIndex(exp[i])
    **if** $p$.children.get(nextindex) is null **then**
      $c$ = $c$.addChild(nextindex)
      $c$.setLabel(exp[i])
    **else**
      $c$ = $c$.children.get(nextindex)
    **end if**
  **end for**
**end for**
```

The visualiser is perhaps best used interactively. Keystroke combinations allow the user to zoom in on specific locations within the program space, and move around to better understand how the algorithm under scrutiny works.

We have implemented our system using Java[32] and the Java Swing [33] two-dimensional graphical library. The operations we use to construct the tree visualisers could however be implemented with any modern graphical system. Java and Swing are convenient portable systems that can be easily attached to our framework and set up with simple graphical programmatic utilities.

4. Visualisation Results

We present a number of visual frames of various algorithms along with a commentary on what convergence

Algorithm 2 Drawing the tree-based data-structure to the screen recursively.

```
with $m$ symbols per program

render(top-level)

FUNCTION render(c)
for $i = 0$ to $linecount$ do
   lines[i].paint()
end for
if children is not null then
   vector2d mystart = getMyStart();
   vector2d myend = getMyEnd();
   if orientation == Horizontal then
      drawDivisionsHorizontal(mystart,myend)
   else
      drawDivisionsVertical(mystart,myend)
   end if
   for $j = 0$ to childrencount do
      render(children[j])
   end for
else
   drawPoint(mycentre)
end if
END
```

actions that are visible. In particular, we now compare the characteristics of the K-GP and K-GPSO in terms of convergence. Figure 3 show successive generations of the K-GP. These figures show that the K-GP is very effective at maintaining diversity. This will become more clear when we discuss the K-GPSO.

Figure 4 shows the second frame of a sample generation in the K-GPSO optimiser. Immediate impressions that this image conveys is the clear use of a global optimum which is used in crossover. It also indicates that there may be an issue in population diversity. In [26], we discussed the parameters ϕ_p, ϕ_g and ω, and mentioned that they are best set to static empirically obtained values. This is as opposed to weighted values depending on the fitness values associated with the *gBest*, *lBest*, and current candidates. The problem with the latter is it is very common for the fitness distance between any candidate and the global best to be disproportionately high. This would cause the crossover point to be chosen so that it is simply the entire *gBest* candidate being replicated.

To make this more clear, we show a plot of the unique candidates by generation for the sample run in Figure 5. Having a good number of unique programs is important to ensure adequate diversity for future crossover operations. The difference in diversity by generation for the K-GP and K-GPSO algorithms is conclusive. We believe that an improvement upon diversity statistics in the K-GPSO would bring about a better convergence rate.

After observing the scores from the sample generation of the K-GPSO, a large number of the programs obtained a score of zero. Essentially, in the flow of the algorithm with score-weighted crossover, this would result in a replication of the global best. Ideally what is necessary, is a higher mutation rate.

Firstly, we adopted a much higher mutation rate of 0.3, (as opposed to 0.1), which did not improve the convergence of the algorithm. The standard deviation of the results was too high to be considered a reliable optimiser. Unique diversity in the population was not maintained, since 0.3 was still too low. The problem with increasing mutation probability further, is that the algorithm would fail to converge at all, as the better solutions would almost certainly be mutated to lower fitness values.

We then experimented with lowering the crossover rate. This was more fruitful, and resulted in a much lower standard deviation among average mean fitness values. A crossover probability of 0.1 seemed to improve the convergence rate. A crossover rate this low does not perform well for genetic algorithms, however. Figure 6 shows frame 2 of a sample generation with this modification. In comparison to Figure 4, what is clear is that most of the population remains stationary. The reason why this performs better, we believe, is due to the more paced movement of particles towards the global best. It is also possible that this K-GPSO algorithm is simply not well suited to this objective function, especially considering that there is some error associated with the function itself.

Figure 7 conveys a sense of how the visualiser might respond to interaction. The top-level program space is shown on the left (generation 100 of a sample run of K-GPSO), and successive zooming in on the area where the most candidate programs are quickly indicates the global best without a doubt. A subtle feature of this is that the lines indicate both a previous program, and a succeeding program. The previous program is represented by a grey dot, whereas the new program is a blue dot. This does add an indication of movement about the global optimum.

5. Discussion

A number of observations on algorithmic behaviours can be made from the visual renderings we obtained.

Most of the insights we obtained from the visualiser seems to give more and more credit to Poli and McPhee's concept of Homologous Crossover [34], where crossover preserves information already shared between candidates. The problem we observe with the K-GPSO is that the global best weighted score is often so great in comparison, that it is simply duplicated.

While the visualisation itself does assist in a qualitative manner, it is far more useful when used interactively. Zooming and movement across the program space is very useful,

Fig. 3

Generations 1-4 (top), 5, 10, and 100 (bottom) of a sample run of the K-GP.

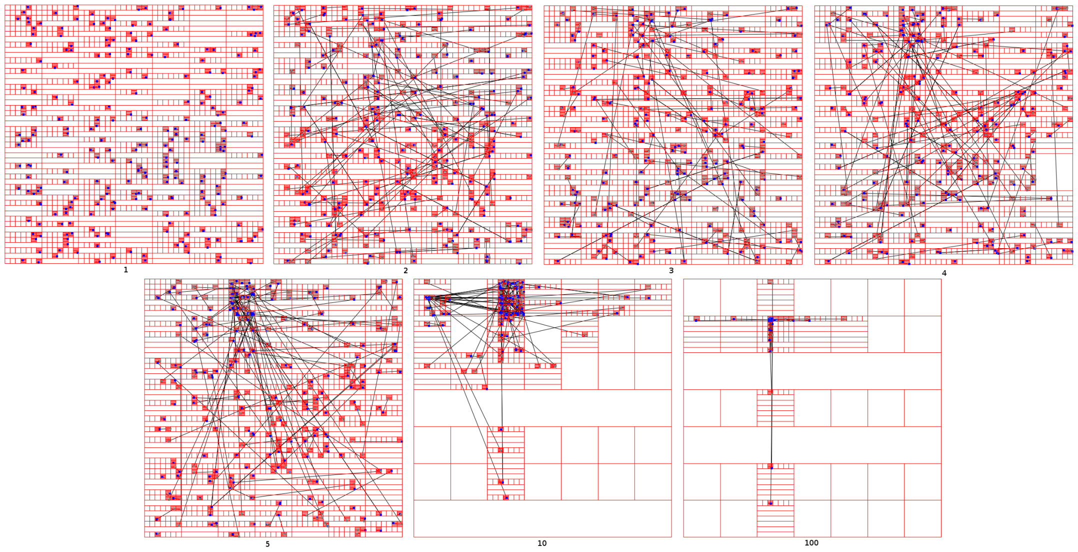


Fig. 4

A visualisation of an early frame of a generation in the K-GPSO optimiser.

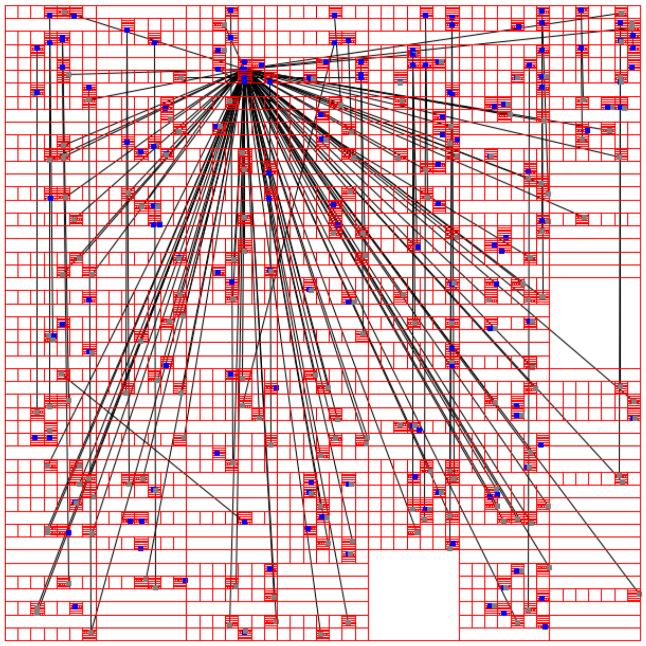
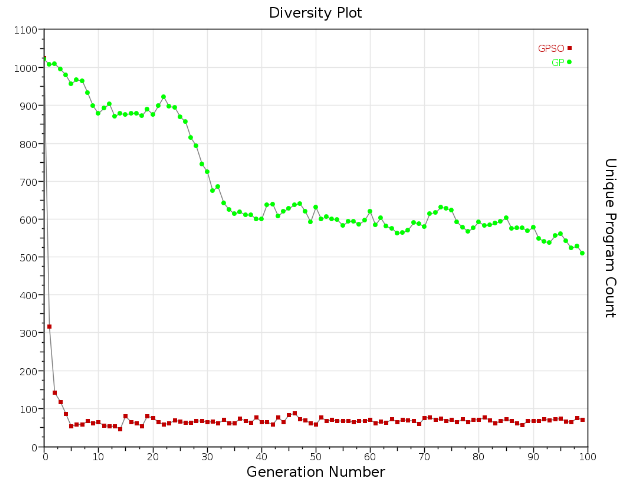


Fig. 5

Diversity plot of the CUDA GP and GPSO algorithms both using K-expressions.



especially for gaining insight into how the algorithm behaves on a microscopic level.

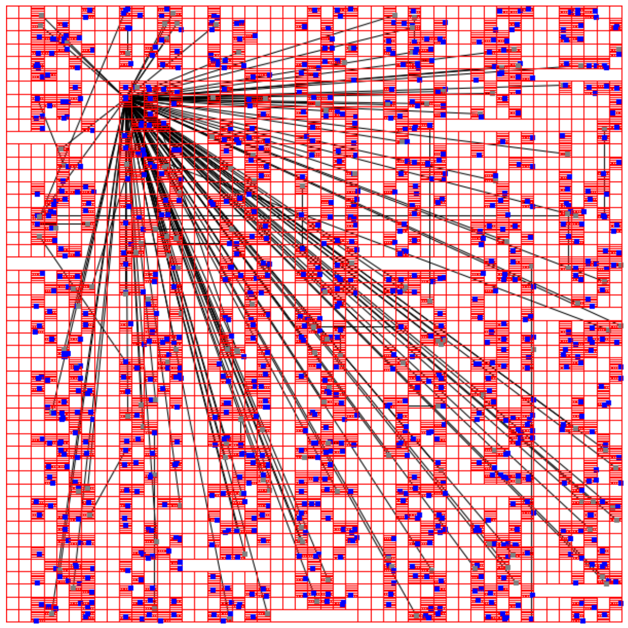
Representing programs in this fashion has some drawbacks however. Spatial distance in the visualisation has no bearing over crossover and mutation operators in their ability to move candidate programs through space. These concepts do not share a similar concept of spatial distance to that of the visualiser. This can result in a more difficult to interpret visualisation at times, as crossover and mutation may translate a certain candidate very far away from the original, while the program may only differ in one symbol.

Indicating movement through this program space for the K-GP (Karva Genetic Programming) algorithm is less meaningful than for the K-GPSO (Karva Geometric Particle Swarm Optimiser) . The reason for this is in the implementation of tournament selection, where, depending on the outcome of the two tournaments, the candidates used in the end may be unrelated to the originals chosen.

Nevertheless, the use of this visualisation has led us to

Fig. 6

VISUALISATION OF FRAME 2 OF THE K-GPSO WITH MODIFIED PARAMETERS, AT THIS FRAME, 907 UNIQUE PROGRAMS ARE PRESENT.



identify what we believe to be the main problem underlying the K-GPSO. The lack of program diversity in this algorithm, especially using weighted scores for multi-parent crossover, results in a great diversity deficiency. Our efforts to correct the K-GPSO saw limited success. From these observations, it seemed that using static values for the parameters in the K-GPSO is not conducive to avoiding local minima issues. Using weighted values according to scores does bring a limited improvement.

6. Conclusions and Future Work

In summary, we have presented an effective visualisation technique for Genetic Programming and its variants. We applied this to our K-GP (Karva Genetic Programming) and K-GPSO (Karva Geometric Particle Swarm Optimiser) algorithms and discussed the merits of this visualisation, and we also presented various modifications to these algorithms inspired from visual cues.

In the past, abstract syntax trees have mainly been analysed using quantitative methods. Visualisations were mostly restricted to the objective function itself, which does give limited information regarding the relative efficacy of candidate programs. We believe that a visualisation such as this gives effective visual cues that inspire improvements.

We have been able to make a number of qualitative observations concerning the algorithms under study by spotting emergent patterns and following visual cues that a interactive human user can readily make, but which would be hard to easily encode a supervisory pattern recognition program to identify. This emphasises the importance of a human-guided optimizer, implemented to work in near real-time.

We anticipate that future work could involve using Graphical Processing Units to further speed the process of rendering images, so that it can be used in real time. It could also be very beneficial to build in landscape analysis to this visualiser to perhaps produce a colour-coded image indicating higher fitness values, or even emit a 3D plot of the landscape itself. There is also scope for rendering trees in three dimensions. Generally 3D rendering is more expensive in terms of computational cost but potentially can pack more and more complex information onto a rendering for a human to spot patterns and changes.

References

[1] William J. Cook, William H. Cunningham, and Alexander Schrijver. *Combinatorial Optimization*. Wiley, 1997. ISBN 0-471-55894-X.

[2] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Applied Mathematics. Princeton, 2006. ISBN 978-0-691-12993-8.

[3] G. V. Wilson and G. S. Pawley. On the stability of the travelling salesman problem algorithm of Hopfield and Tank. *Biol. Cybern.*, 58:63–70, 1988.

[4] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[5] Robert Axelrod. The emergence of cooperation among egoists. *The American Political Science Review*, 75:306–318, 1981.

[6] Kennedy and Eberhart. Particle swarm optimization. *Proc. IEEE Int. Conf. on Neural Networks*, 4:1942–1948, 1995.

[7] John R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, June 1994.

[8] J. H. Holland. *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press, 1975.

[9] R. Poli, W.B. Langdon, and N.F. McPhee. *A field guide to genetic programming*. lulu.com, 2008.

[10] Sean Luke. Genetic programming produced competitive soccer softbot teams for robocup97. In J. R. Koza, W. Banzhaf, K. Chellapilla, D. Kumar, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the 3rd annual conference*, pages 214–222. Morgan Kaufmann, San Mateo, California, 1998.

[11] Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Co-evolving soccer softbot team coordination with genetic programming. *Robocup-97: Robot soccer world cup I*, 1:398–411, 1998.

[12] Vladan Babovic and Maarten Keijzer. Genetic programming as a model induction engine. *Journal of Hydroinformatics*, 2(1):35–60, 2000.

[13] Mark Crosbie and Eugene H. Spafford. Applying genetic programming to intrusion detection. Technical report, Department of Computer Sciences, Purdue University, West Lafayette, 1995. AAAI Technical Report FS-95-01.

[14] Steven M. Manson. Agent-based modeling and genetic programming for modeling land change in the southern yucatán peninsular region of mexico. *Agriculture Ecosystems & Environment*, 111:47–62, 2005.

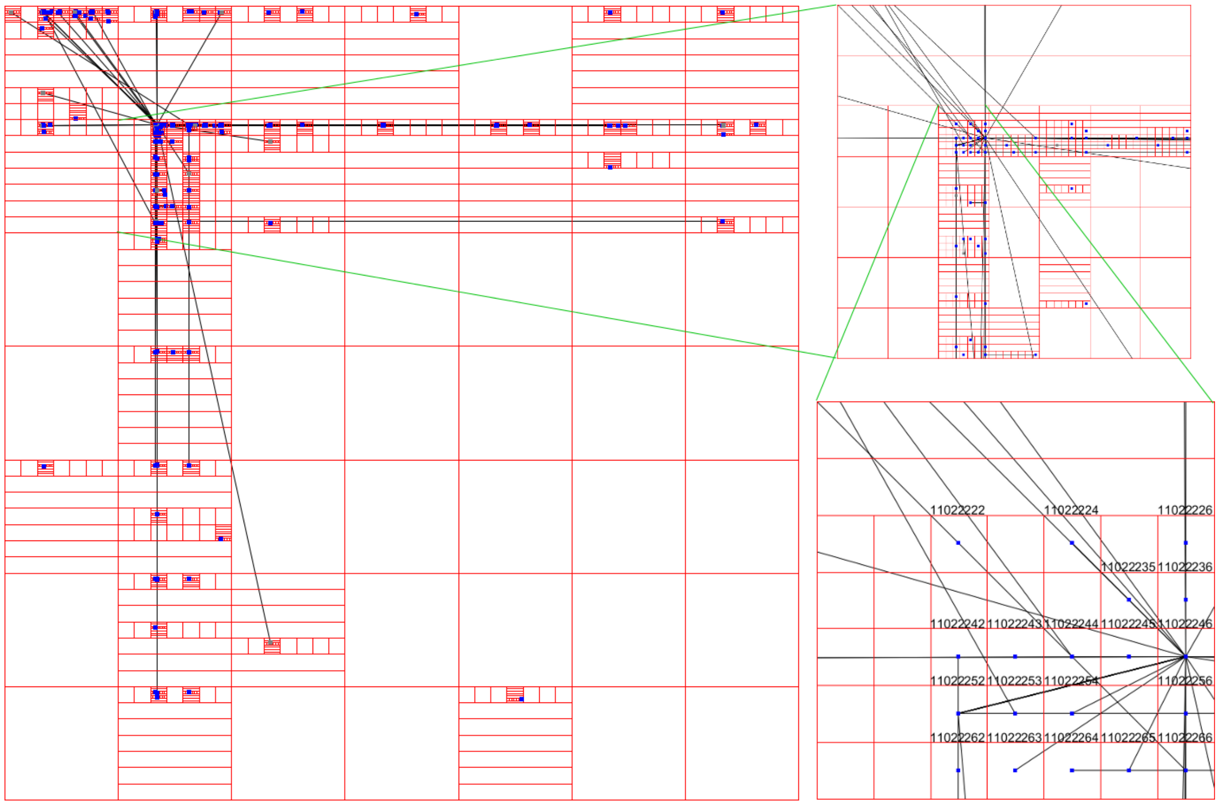
[15] Riccardo Poli and Stefano Cagnoni. Genetic programming with user-driven selection: Experiments on the evolution of algorithms for image enhancement. In *Genetic Programming 1997: Proceedings of the 2nd Annual Conference*, pages 269–277. Morgan Kaufmann, 1997.

[16] Julian F. Miller and Stephen L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.

[17] Simon L. Harding and Wolfgang Banzhaf. Distributed genetic programming on gpus using cuda. Submitted to Genetic Programming and Evolvable Machines, 2009.

Fig. 7

SUCCESSIVE ZOOMING TO THE GLOBAL OPTIMUM'S LOCATION.



[18] Leandro Cupertino and Cristiana Bentes. Evolving cuda ptx programs by quantum inspired linear genetic programming. In *Proceedings of GECCO'11*, 2011.

[19] Tom Castle and Colin G. Johnson. Evolving high-level imperative program trees with strongly formed genetic programming. In *Proceedings of the 15th European Conference on Genetic Programming, EuroGP*, volume 7244, pages 1–12. Springer, April 2012.

[20] W. B. Langdon. A many-threaded cuda interpreter for genetic programming. In Ana Isabel Esparcia-Alcazar, Aniko Ekart, Sara Silva, Stephen Dignum, and A. Sima Uyar, editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP*, pages 146–158. Springer, April 2010.

[21] W. B. Langdon and Wolfgang Banzhaf. A simd interpreter for genetic programming on gpu graphics cards. In M. O'Neill, L. Vanneschi, A.I. Esparcia, and S. Gustafson, editors, *Proceedings of the 11th European Conference on Genetic Programming, EuroGP*, March 2008.

[22] Kenneth E. Kinnear, Jr. Fitness landscapes and difficulty in genetic programming. In *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, volume 1, pages 142–147, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[23] Steven M. Gustafson. *An Analysis of Diversity in Genetic Programming*. PhD thesis, School of Computer Science, University of Nottingham, England, 2004.

[24] Cândida Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129, 2001.

[25] Alwyn V. Husselmann and K. A. Hawick. Genetic programming using the karva gene expression language on graphical processing units. In *Proc. 10th International Conference on Genetic and Evolutionary Methods (GEM'13)*, number CSTN-171, page GEM2456. WorldComp, 22-25 July 2013.

[26] Alwyn V. Husselmann and K. A. Hawick. Geometric optimisation using karva for graphical processing units. In *Proc. 15th International Conference on Artificial Intelligence (ICAI'13)*, number CSTN-191, page ICA2335, Las Vegas, USA, 22-25 July 2013. WorldComp.

[27] Julian Togelius, Renzo De Nardi, and Alberto Moraglio. Geometric pso + gp = particle swarm programming. In *2008 IEEE Congress on Evolutionary computation (CEC 2008)*, 2008.

[28] A. Moraglio. *Towards a Geometric Unification of Evolutionary Algorithms*. PhD thesis, Computer Science and Electronic Engineering, University of Essex, 2007.

[29] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1:33–57, 2007.

[30] Riccardo Poli, Leonardo Vanneschi, William B. Langdon, and Nicholas Freitag McPhee. Theoretical results in genetic programming: the next ten years? *Genetic Programming and Evolvable Machines*, 11:285–320, 2010.

[31] Arno Leist, Daniel P. Playne, and K. A. Hawick. Exploiting Graphical Processing Units for Data-Parallel Scientific Applications. *Concurrency and Computation: Practice and Experience*, 21(18):2400–2437, 25 December 2009. CSTN-065.

[32] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. JavaSoft Series. Addison Wesley Longman, 1996. ISBN 0-201-63451-1.

[33] Marc Hoy, Dave Wood, Marc Loy, James Elliot, and Robert Eckstein. *Java Swing*. O'Reilly and Associates, 2002. ISBN:0596004087.

[34] Riccardo Poli and Nicholas F. McPhee. Exact schema theory for gp and variable-length gas with homologous crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 2001.