Solving the 15-Puzzle

Alan Turing and **John von Neumann**
`{alturing, jovonneumann}@davidson.edu`
Davidson College
Davidson, NC 28035
U.S.A.

Abstract

The abstract should be between four and seven sentences long. Introduce the problem you are studying. Describe what you did. Summarize your results — what did you discover, what is the main take-away message?) Basically, you're trying to sell your paper to the reader, so be brief and to the point. **Do *not* include any citations in the abstract.**

1 Introduction

In this section, you should introduce the reader to the problem you are attempting to solve. For example, for the first project: describe the 15-puzzle, and why it's interesting as an A.I. problem. You should also cite and briefly describe other related papers that have tackled this problem in the past — things that came up during the course of your research. In the AAAI style, citations look like (**?**) (see the comments in the source file `intro.tex` to see how this citation was produced). Conclude by summarizing how the remainder of the paper is organized.

Overall, the aim in this section is context-setting: what is the big-picture surrounding the problem you are tackling here?

2 Background

Describe any background information that the reader would need to know to understand your work. You do not have to explain algorithms or ideas that we have seen in class. Rather, use this section to describe techniques that you found elsewhere in the course of your research, that you have decided to bring to bear on the problem at hand. Don't go overboard here — if what you're doing is quite detailed, it's often more helpful to give a sketch of the big ideas of the approaches that you will be using. You can then say something like "the reader is referred to X for a more in-depth description of...", and include a citation.

Alternately, you may have designed a novel approach for the problem — your own algorithm or heuristic, say. A description of these would also be placed in this section (use subsections to better organize the content in this case).

Enumerating

Create bulleted lists by using the `itemize` command (see source code):

- Item 1
- Item 2
- Item 3

Create numbered lists by using the `enumerate` command (see source code):

1. Item 1
2. Item 2
   (a) Sub-item 2a
   (b) Sub-item 2b
3. Item 3

Formatting Mathematics

Entire books have been written about typesetting mathematics in LaTeX , so this guide will barely scratch the surface of what's possible. But it contains enough information to get you started, with pointers to resources where you can learn more. First, the basics: all mathematical content needs to be written in "math-mode" — this is done by enclosing the content within $ symbols. For example, the code to produce $6x + 2 = 8$ is `$6x + 2 = 8$`. Note that this is only good for in-line math; if you would like some stand-alone math on a separate line, use *two* $ symbols. For example, `$$6x + 2 = 8$$` produces:

$$6x + 2 = 8$$

Here are various other useful mathematical symbols and notations — see the source code to see how to produce them.

- Sub- and super-scripts: $e^x, a_n, e^{2x+1}, a_{n+2}, f^i_{n+1}$
- Common functions: $\log x, \sin x$
- Greek symbols: $\epsilon, \phi, \pi, \Pi, \Phi$
- Summations: $\sum_{i=0}^{i=100} i^2$

- Products: $\prod_{i=0}^{\infty} 2^{-i}$
- Fractions: $3/2$

$$\frac{x+5}{2\cdot\pi}$$

Other useful resources:

- Find the LaTeX command you're looking for by drawing what you want to produce[1]:`http://detexify.kirelabs.org/classify.html`
- Ask others: `http://tex.stackexchange.com/`
- Every LaTeX symbol ever: `http://tinyurl.com/6s85po`

3 Experiments

This experiment was designed to use a genetic algorithm to adapt a population of expression trees to correctly identify two unknown functions. In separate runs, populations of 1000 expression trees were adapted through 50 generations, with the goal of matching the goal functions.

Genetic crossover was achieved by probabilistically combining expression trees based on their fitness functions. The fitness score of each tree is calculated as the reciprocal of the total error between the expression tree and goal function, evaluated at a large number of points. The reciprocal is taken in order to give more accurate trees, with lower sum errors, higher fitness scores. Trees were then chosen randomly according to a probabilistic distribution weighted by the fitness scores, with trees replaced back into the population after each crossover.

One of the main concepts behind genetic algorithms is that parents in the algorithm pass on traits recognizably to children. By choosing the most fit expression trees to reproduce, the algorithm should, after a number of generations, converge on a locally optimal solution. In order for preferable characteristics from parent expression trees to be transferred to child trees, the crossover operation chose a random node from each tree, then swapped the subtrees extending from the chosen nodes. Child trees therefore were composed of partial sections of each of the parent trees. Because trees were chosen with replacement, and combine randomly during each crossover, populations may have multiple identical trees or trees made by combining the same two parent trees at different nodes.

After each crossover operation, a portion of the population was mutated in order to introduce random variation. For a given expression tree, the mutation operation randomly selected a node and altered the value or mathematical operation at that node based on the original value. To avoid damaging the behavior of the tree too much, thereby potentially lowering the accuracy of the algorithm, the mutated operation or value was chosen to structurally match the original node. Leaf nodes were only replaced with terminals: randomly chosen constants or appropriate variables. Nodes originally containing binary operations ($+$, $-$, $\times$, $/$) were only replaced with other operations of that nature.

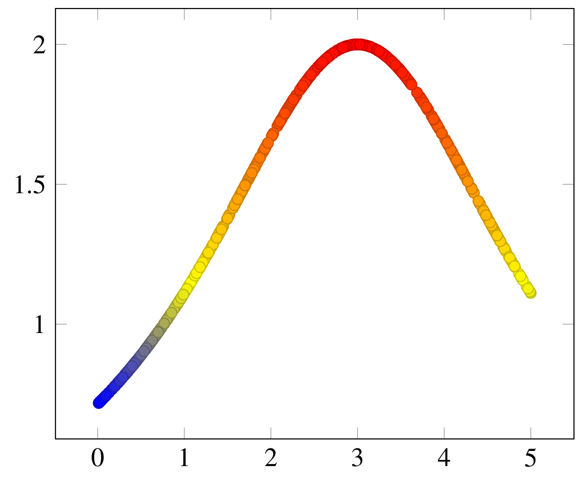

Figure 1: Non-zero section of goal function 1

It is of note that none of the nodes of the expression trees contain the power operation, e.g. x^2. This was a code optimization choice made because power operations can be created through multiplication of appropriate variables. Removing power operations decreased the chance that an expression tree would become arbitrarily complicated through high powers, instead favoring low integer powers – x, x^2, x^3.

4 Results

In order to help the algorithm converge on the correct solutions, the parameters guiding the behavior of the algorithm were tailored to increase population diversity, maintain successful trees, and correctly adapt to the goal functions. The sampling range of the functions also had significant impact on the algorithm's accuracy. When comparing an expression tree to the goal functions, which determined the fitness of the trees, best results were seen when the error was calculated over a range where the goal function was significantly non-zero. If the fitness was calculated over larger ranges of input values, the expression trees tended strongly towards constant expressions that minimize total error but describe only the asymptotic tails of the function.

Data points taken from the first goal function are displayed in Table 1. The second goal function is four dimensional, and cannot be conveniently displayed.

[1] Thanks to Dr. Kate Thompson for pointing me to this resource. Also, this is how you create a footnote. Also, don't overuse them — prefer citations and use the acknowledgements section when possible. I usually only use footnotes when I want to link to include a pointer to a web site.

Despite careful adjustment of parameters governing tree construction, mutation rate, and the sampling range of the fitness function, we were unable to develop any non-constant expressions for the goal functions as given. While closing the sampling range to only include the area of the fitness function displaying interesting, non-asymptotic behavior did result in lower error amongst the most accurate trees, the genetic algorithm was unable to determine a more accurate function.

In order to attempt to match the first goal function, we inverted the output of the function in order to alter the expression needed to produce it. Surprisingly, this met with success, as the algorithm consistently finds x^2 solutions which closely approximate the goal function. Unlike the runs with the original goal function, the fitness scores converge when evaluating based on the inverted function. Through this technique, we were able to arrive at an accurate function, which we then invert to conclude that the first goal function is

$$y = \frac{10}{x^2 - 6x + 14}$$

Unfortunately, this technique has not been successful for the second goal function, which has 3 input variables. For that reason, we were also unable to plot the data in order to determine an appropriate sampling region or to visualize the function in general. When attempting to fit the second goal function, our algorithm always relies on a constant function. Unfortunately, this is clearly not a correct solution, as the error associated with such trees is massive.

5 Conclusions

In this section, briefly summarize your paper — what problem did you start out to study, and what did you find? What is the key result / take-away message? It's also traditional to suggest one or two avenues for further work, but this is optional.

6 Acknowledgements

This section is optional. But if there are people you'd like to thank for their help with the project — a person who contributed some insight, friends who volunteered to help out with data collection, etc. — then this is the place to thank them. Keep it short!