

Literature Review of Optimisation Techniques to the Graph Colouring Problem

Scott Whitemore & Brendon Veronese

October 19, 2015

Abstract

This is an abstract.

Chapter 1

Literature Review

1.1 Introduction

1.2 Johnson - Simulated Annealing

1.3 Gravitational Swarm Intelligence

1.4 Genetic Algorithm

1.5 Flower Pollination

Chapter 2

Implementations & Original Contributions

2.1 Introduction

2.2 Random Brute Buckets

The Random Brute Buckets (RBB) algorithm presented here is a class of Successive Augmentation Technique. It is at its core an implementation of an insertion sort algorithm which samples vertices from the graph in a random order. This algorithm is guaranteed to find a valid colouring for graphs that are directed and undirected. It is however not bounded, and there is no guarantee that the algorithm will converge upon the best colouring.

2.2.1 Implementation and Problems

The RBB algorithm is presented in Algorithm 1 below.

Algorithm 1 Random Brute Buckets

```
1: procedure SOLVE(Graph  $g$ , long  $iterationLimit$ ) ▷  $g$  is predefined
2:    $currentIteration \leftarrow 0$ 
3:    $currentBestColouring \leftarrow \infty$ 
4:   Create empty list of buckets  $bList$ 
5:   while  $currentIteration < iterationLimit$  do
6:     Populate vertex set  $V$ 
7:     while  $V \neq \emptyset$  &&  $|bList| < currentBestColouring$  do
8:        $v \leftarrow$  random vertex from  $V$  ▷ Remove  $v$  from  $V$ 
9:        $vertexPlaced \leftarrow FALSE$ 
10:      for all  $bucket \in bList$  do
11:        if  $bucket$  contains no conflicts with  $v$  then
12:          add  $v$  to  $bucket$ 
13:           $vertexPlaced \leftarrow TRUE$ 
14:          break for
15:        else continue
16:      end if
17:    end for
18:    if  $!vertexPlaced$  then ▷ create a place for the vertex
19:      create new bucket  $b_0$ 
20:      add  $v$  to  $b_0$ 
21:      add  $b_0$  to  $bList$ 
22:    end if
23:  end while
24:  if  $|b| < currentBestColouring$  then
25:     $currentBestColouring = |bList|$ 
26:  end if
27:   $currentIteration ++$ 
28: end while
29: return  $currentBestColouring$ 
30: end procedure
```

This algorithm always improves upon the best colouring (which is initially set to ∞ on line 3) in the first iteration, and upon subsequent iterations, if a better solution is stumbled upon randomly, that solution is set as the current best solution. The algorithm returns the best solution found after a set number of iterations set at runtime.

2.2.2 Workarounds

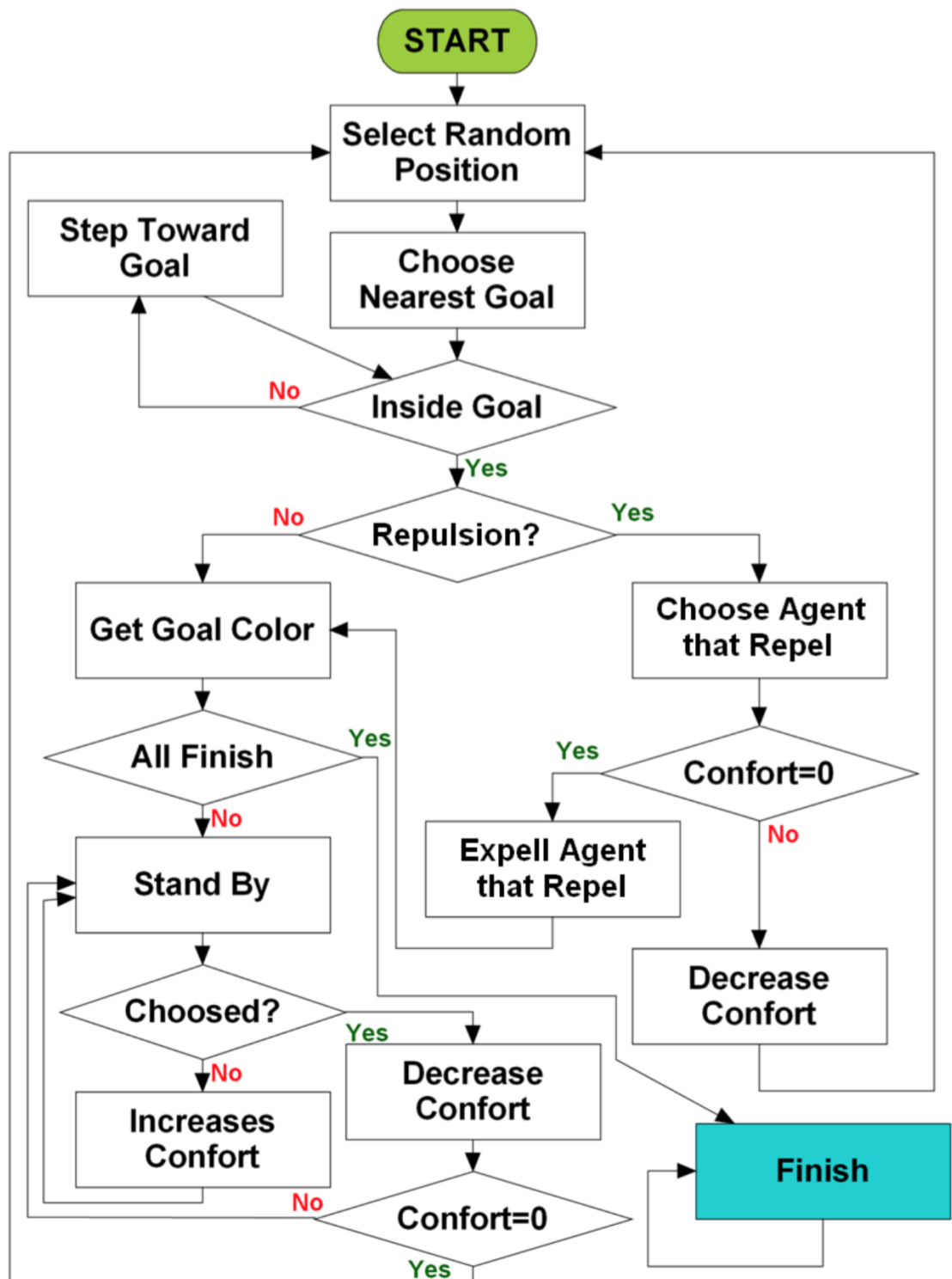
This algorithm is very fast at finding initial solutions. This algorithm does not try to improve upon solutions that it finds, so local minima offer no resistance to finding the solution. This algorithm is not guaranteed to find the best solution. Indeed even after infinite attempts, there is no guarantee that the best solution will be found. That being said, the algorithm is generally able to find solutions close to the optimal colouring for most graphs extremely quickly and with a minimum use of resources (compared to other algorithms presented in this paper).

2.3 Gravitational Swarm Intelligence

The Gravitational Swarm Intelligence Algorithm presented in this paper is a modified version of the algorithm presented in the paper "Gravitational Swarm for Graph Coloring" by Israel Carlos Rebollo Ruiz [1]. Swarm Intelligence is a model where the emergent collective behavior is the outcome of a process of self-organization, where the agents evolve autonomously following a set of internal rules for their motion, interaction with the environment and the other agents. In this algorithm, the agents are subjected to an environment subject to a version of Newtonian Gravity. Centres of attraction (wells) are placed in the environment and the agents move based upon their attraction to these wells. Each well represents a distinct grouping or colouring of the underlying graph. Agents experience a repulsive force if they try and enter a well already containing agents that prohibit a valid colouring. If an agent enters a well that is not prohibited, then it stops moving and takes on that well's colour. The algorithm ends once a set iteration limit is reached, or all agents settle into the wells, and hence a valid colouring is found.

2.3.1 Implementation and Problems

The action of each is presented below:



Each agent moves in the toric world until all agents fall into the "Stand By" State. Then once the last agent triggers the "All Finished" State, the current colouring is accepted as valid. This algorithm presented some issues when implemented for testing. The comfort statistic that each agent tracks can grow unboundedly if the agents repeatedly fall into wells that they are repulsed from. This causes all agents to grow in comfort, making the local minima harder to improve with every iteration and the chance of jumping out and finding a valid solution fall dramatically.

2.3.2 Workarounds

2.3.3 Additions

2.4 Genetic Algorithm

The Genetic Algorithm presented here is a variant of the algorithm described by [2]. The algorithm was implemented as described, then tweaked to improve efficiency and to experiment with parameter tweaking and the effect of changing certain flows within the algorithm. These changes are discussed further in the Additions (2.4.3) section.

2.4.1 Implementation and Problems

The algorithm is implemented as follows:

Algorithm 2 Genetic Algorithm with Wisdom of Crowds

```

1: procedure SOLVE(Graph  $g$ ,  $iterationLimit$ ,  $numChromosomes$ )  $\triangleright g$  is predefined
2:    $currentAttempt \leftarrow 0$ 
3:    $currentBestColouring \leftarrow \Delta(g) + 1$ 
4:    $aggregateChromosome \leftarrow$  chromosome of randomly assigned colours.
5:   while  $currentIteration < iterationLimit$  do
6:      $population \leftarrow$  set of chromosomes with randomly assigned colours. (up to  $numColours$ )
7:     if solvePop() then
8:        $aggregateChromosome \leftarrow$  chromosome of randomly assigned colours.
9:        $currentAttempt \leftarrow 0$ 
10:    else
11:       $currentAttempt \leftarrow currentAttempt + 1$ 
12:    end if
13:  end while
14:  return  $currentBestColouring$ 
15: end procedure

```

Algorithm 3 Genetic Algorithm with Wisdom of Crowds - Tick Generation

```
1: procedure SOLVEPOP(Graph  $g$ ,  $iterationLimit$ ,  $numChromosomes$ )  $\triangleright g$  is predefined
2:    $currentIteration \leftarrow 0$ 
3:   while  $currentIteration < iterationLimit$  and best solution has cost  $> 0$  do
4:      $currentIteration \leftarrow currentIteration + 1$ 
5:     if best chromosome has cost  $\geq altMethodThreshold$  then
6:        $parents \leftarrow getParentsA()$ 
7:     else
8:        $parents \leftarrow getParentsB()$ 
9:     end if
10:     $child \leftarrow crossOver(parents)$ 
11:    if  $rand < mutChance$  then
12:      if best chromosome has cost  $\geq altMethodThreshold$  then
13:         $child \leftarrow mutateA()$ 
14:      else
15:         $child \leftarrow mutateB()$ 
16:      end if
17:    end if
18:    add  $child$  to  $population$ 
19:    remove bottom performing half of population
20:    repopulate up to  $numChromosomes$ 
21:  end while
22:  if  $currentIteration \geq iterationLimit$  then
23:    perform  $wisdomOfCrowds()$   $\triangleright$  generate  $aggregateChromosome$  by voting
24:    add  $aggregateChromosome$  to population
25:  end if
26:  if best solution has cost 0 then
27:     $currentBestSolution \leftarrow currentBestSolution - 1$ 
28:    return true
29:  else
30:    return false
31:  end if
32: end procedure
```

Algorithm 4 Genetic Algorithm with Wisdom of Crowds - Parent Selection

```
1: procedure GETPARENTSA
2:    $tempParents \leftarrow$  choose two random chromosomes from population.
3:    $parent1 \leftarrow$  fitter of  $tempParents$ 
4:    $tempParents \leftarrow$  choose two random chromosomes from population.
5:    $parent2 \leftarrow$  fitter of  $tempParents$ 
6:   return  $parent1, parent2$ 
7: end procedure
8:
1: procedure GETPARENTSB
2:   return top performing chromosome, top performing chromosome
3: end procedure
```

Algorithm 5 Genetic Algorithm with Wisdom of Crowds - Crossover

```
1: procedure CROSSOVER
2:    $child \leftarrow$  colours up to and including a random point from  $parent1$ , followed by the colours
   from  $parent2$  from that point on in the chromosome.
3:   return child
4: end procedure
```

Algorithm 6 Genetic Algorithm with Wisdom of Crowds - Child Mutation

```
1: procedure MUTATEA
2:   for all  $vertex$  in chromosome do
3:     if  $vertex$  has a conflict then
4:        $adjacentColours \leftarrow$  all adjacent colours to  $vertex$ 
5:        $validColours \leftarrow allColours - adjacentColours$ 
6:        $newColour \leftarrow$  random colour from  $validColours$ 
7:       set chromosome colour at  $vertex$  to be  $newColour$ 
8:     end if
9:   end for
10: end procedure
11:
1: procedure MUTATEB
2:   for all  $vertex$  in chromosome do
3:     if  $vertex$  has a conflict then
4:        $newColour \leftarrow$  random colour from allColours
5:       set chromosome colour at  $vertex$  to be  $newColour$ 
6:     end if
7:   end for
8: end procedure
```

Algorithm 7 Genetic Algorithm with Wisdom of Crowds - Wisdom Of Artificial Crowds

```
1: procedure WISDOMOFCROWDS
2:    $expertChromosomes \leftarrow$  best half of final population
3:    $aggregateChromosome \leftarrow$  best performing chromosome
4:   for all  $vertex$  do
5:     if  $vertex$  is part of a bad edge then
6:        $newColour \leftarrow$  the most used colour for  $vertex$  among  $expertChromosomes$ 
7:       set colour at  $vertex$  of  $aggregateChromosome$  to be  $newColour$ 
8:     end if
9:   end for
10: end procedure
```

2.4.2 Workarounds

The algorithm provides some parameters to tailor the solving method to be able to deal with local optima in several ways. Firstly there are 2 different parent selection methods that are used depending upon how close the best chromosome is to a valid colouring. Secondly there are two different mutation

methods that are used depending upon how close the best chromosome is to a valid colouring. It was found that while these methods allowed for some level of control over the algorithm, it was by no means as configurable as the other algorithms that were implemented. To remedy this and to produce our own variant of the Genetic Algorithm for solving the GCP, alterations were made to the flow to allow more control over the algorithm.

2.4.3 Additions

The first deviation to the documented algorithm was to the parent selection criterion. It was found that having the best solution be the only parent that is mutated when it is sufficiently close to a valid colouring slowed the convergence rate of the other chromosomes to zero. To remedy this, we implemented a switch such that a certain percent of the time, the parents chosen were the best chromosome and itself. But a small percentage of the time, the parents would be the best chromosome and a random chromosome from the population. This prevented the stagnation of the problem, and allowed for other changes to be made to improve the algorithm. The second part of the algorithm identified for change was the crossover method. This method originally selected the first portion of *parent1* and mixed it with the complement of that section of *parent2*. This process was changed to a simple 50/50, such that the chance of the first section coming from *parent1* was equally weighted to the first section coming from *parent2*. This prevented possible bad colourings from being propagated simply because they occurred early in the chromosome. The final change to the algorithm was in response to results collected from the algorithm. It was found the wisdom of crowds voting mechanism was finding the solution far more successfully than the Genetic Algorithm itself. This prompted a change to the algorithm whereby a vote could not take place before certain conditions were met in the agreement of the chromosomes. This led to a much higher runtime for the algorithm, but once a vote could commence, the algorithm was much more likely to either improve the colouring or produce a chromosome with many fewer conflicts. This method was settled upon after several other methods were trialed and discarded. As the algorithm searches for a tighter colouring, it takes many more generations for the chromosomes to converge. This method was settled upon since it allowed the algorithm to be applied to any graph with a minimum of tweaking, yet a valid vote will always take place, since the algorithm will always reach a point of agreement, even if the solution is not valid for the given graph. Setting the level of agreement that each chromosome must exhibit allows the algorithm to be sped up or slowed down. If the limit is relaxed, each trial takes less time to complete, yet the wisdom of crowds vote has a lower chance of improving the solution. Tightening the bounds has the effect of dramatically increasing the runtime of the algorithm but increases the chance that the wisdom of crowds vote will yield a better solution. Setting the bound too tight however will result in the algorithm running for an infinite amount of time (if the level of agreement specified is not possible). The last change that was made to this algorithm was to create a variable parameter for how much of the population was discarded and repopulated at random on each generation. This parameter is responsible for setting how much of the solution space is tried. Each iteration, the random nature of the generation process has the possibility to generate a better solution to the problem. However, the larger the proportion of the population that is discarded upon each iteration, the lower probability that the wisdom of crowds vote that occurs at the end of each trial will improve upon the solution. For this reason, the proportion of the population that is allowed to participate in the vote is set to always be less than (or equal to) the proportion that is kept in each iteration, so that random noise is not added to the voted data when a vote takes place.

2.5 Flower Pollination

2.5.1 Implementation and Problems

2.5.2 Workarounds

2.5.3 Additions

Chapter 3

Results & Discussion

3.1 Algorithms

Bibliography

- [1] Israel Carlos Rebollo Ruiz. *Gravitational Swarm for Graph Coloring*. <http://www.chu.eus/ccwintco/uploads/3/3a/Tesis-Israel-final.pdf>, 2012.
- [2] Musa M. Hindi and Roman V. Yampolskiy. *Genetic Algorithm Applied to the Graph Coloring Problem*. http://ceur-ws.org/Vol-841/submission_10.pdf, 2015.