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 ← 0
3:     currentBestColouring ← ∞
4:     Create empty list of buckets bList
5:     while currentIteration < iterationLimit do
6:         Populate vertex set V
7:         while V ≠ ∅ && |bList| < currentBestColouring do
8:             v ←random vertex from V                           ▷ Remove v from V
9:             vertexPlaced ← FALSE
10:            for all bucket ∈ bList do
11:                if bucket contains no conflicts with v then
12:                    add v to bucket
13:                    vertexPlaced ← 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 b0
20:                add v to b0
21:                add b0 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

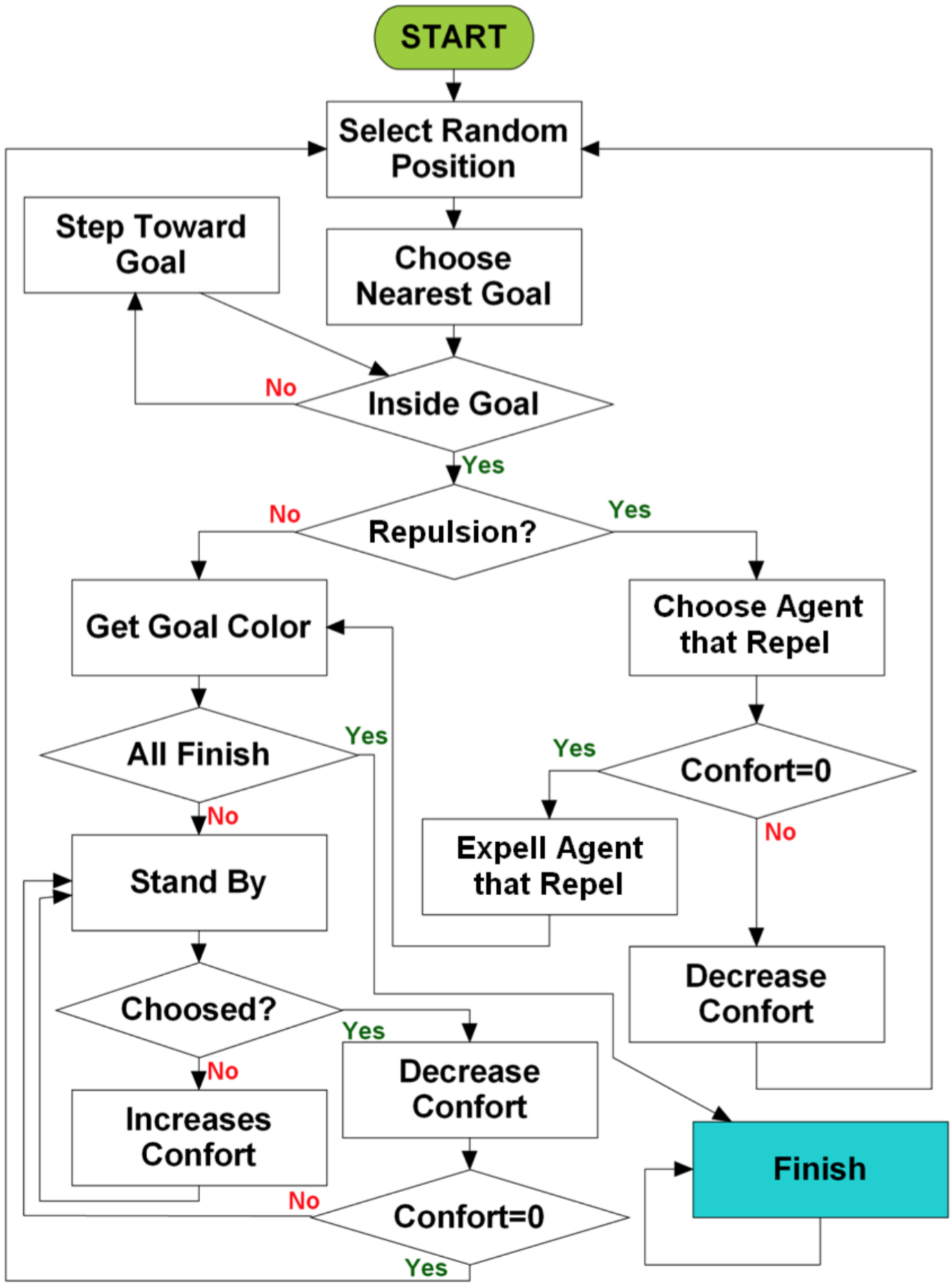
This algorithm is very fast at finding initial solutions. This algorithm does not try to improve upon solutions that it finds, so local minima offer no resistance to finding the solution. This algorithm is not guaranteed to find the best solution. Indeed even after infinite attempts, there is no guarantee 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 underlaying 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:


START
Select Random Position
Step Toward Goal
Choose Nearest Goal
No
Inside Goal
Yes
No
Repulsion?
Yes
Get Goal Color
Choose Agent that Repel
Yes
All Finish
Yes
Confort=0
No
Stand By
Expell Agent that Repel
No
Decrease Confort
Choosed?
Yes
No
Decrease Confort
Increases Confort
Finish
No
Confort=0
Yes

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 minuma 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

```
procedure SOLVE(Graph g, iterationLimit, numChromosomes)        ▷ g is predefined
    currentAttempt ← 0
    currentBestColouring ← Δ(g) + 1
    aggregateChromosome ← chromosome of randomly assigned colours.
    while currentIteration < iterationLimit do
        population ← set of chromosomes with randomly assigned colours. (up to numColours)
        if solvePop() then
            aggregateChromosome ← chromosome of randomly assigned colours.
            currentAttempt ← 0
        else
            currentAttempt ← currentAttempt + 1
        end if
    end while
    return currentBestColouring
end procedure
```

Algorithm 3 Genetic Algorithm with Wisdom of Crowds - Tick Generation

```
procedure solvePop(Graph g, iterationLimit, numChromosomes)          ▷ g is predefined
    currentIteration ← 0
    while currentIteration < iterationLimit and best solution has cost > 0 do
        currentIteration ← currentIteration + 1
        if best chromosome has cost ≥ altMethodThreshold then
            parents ← getParentsA()
        else
            parents ← getParentsB()
        end if
        child ← crossOver(parents)
        if rand < mutChance then
            if best chromosome has cost ≥ altMethodThreshold then
                child ← mutateA()
            else
                child ← mutateB()
            end if
        end if
        add child to population
        remove bottom performing half of population
        repopulate up to numChromosomes
    end while
    if currentIteration ≥ iterationLimit then
        perform wisdomOfCrowds()          ▷ generate aggregateChromosome by voting
        add aggregateChromosome to population
    end if
    if best solution has cost 0 then
        currentBestSolution ← currentBestSolution − 1
        return true
    else
        return false
    end if
end procedure
```

Algorithm 4 Genetic Algorithm with Wisdom of Crowds - Parent Selection

```
procedure getParentsA
    tempParents ← choose two random chromosomes from population.
    parent1 ← fitter of tempParents
    tempParents ← choose two random chromosomes from population.
    parent2 ← fitter of tempParents
    return parent1, parent2
end                                                          procedure

procedure getParentsB
    return top performing chromosome, top performing chromosome
end procedure
```

Algorithm 5 Genetic Algorithm with Wisdom of Crowds - Crossover

```
procedure crossOver
    child ← colours up to and including a random point from parent1, followed by the colours
  from parent2 from that point on in the chromosome.
    return child
end procedure
```

Algorithm 6 Genetic Algorithm with Wisdom of Crowds - Child Mutation

```
procedure MutateA
    for all vertex in chromosome do
        if vertex has a conflict then
            adjacentColours ← all adjacent colours to vertex
            validColours ← allColours - adjacentColours
            newColour ← random colour from validColours
            set chromosome colour at position vertex to be newColour
        end if
    end for
end                                                   procedure

procedure MutateB
    for all vertex in chromosome do
        if vertex has a conflict then
            newColour ← random colour from allColours
            set chromosome colour at position vertex to be newColour
        end if
    end for
end procedure
```

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

```
procedure WisdomOfCrowds
    expertChromosomes ← best half of final population
    aggregateChromosome ← best performing chromosome
    for all vertex dog
        if vertex is part of a bad edge then
            newColour ← the most used colour for vertex among expertChromosomes
            set colour at vertex of aggregateChromosome to be newColour
        end if
    end for
end procedure
```

2.4.2 Workarounds

2.4.3 Additions

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.ehu.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.