THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

SCHOOL OF ENGINEERING

APPLYING SWARM INTELLIGENCE TO SOLVE HEYAWAKE PUZZLES

GREGORY ALAN KRISTON

Spring 2010

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Software Engineering
with honors in Software Engineering

Reviewed and approved* by the following:

Wen-Li Wang
Associate Professor of Electrical and Computer Engineering
Thesis Supervisor and Honors Adviser

Christopher Coulston
Associate Professor of Electrical and Computer Engineering
Faculty Reader

*Signatures on file in the Schreyer Honors College

Abstract

Real world applications have many mutual restricted factors. Heyawake is one paper and pencil logic puzzle published by NIKOLI that has multiple constraints and is NP-complete. A heuristic method for solving the puzzle was developed and demonstrated. The method is based on the ant colony swarm intelligence technique. The developed method uses a two phase approach to handle the multiple constraints placed on the puzzle. The two phase approach demonstrated good performance on puzzles with highly restrictive characteristics and poorer performance on puzzles with less restrictive characteristics.

Contents

Chapter 1

Introduction

Heyawake[1] is one of many NIKOLI pencil and paper logic puzzles gaining popularity throughout the world. Heyawake is one puzzle problem with multiple constraints. The puzzle is NP–complete — i.e. it is difficult for a computer to solve [2]. This puzzle problem is used as an example of how to apply software learning techniques to one problem with multiple constraints.

There are various techniques for software learning, including hill climbing, simulated annealing, swarm intelligence, and neural networks. Most are heuristic techniques — they follow certain rules to search for an acceptable solution, within a reasonable amount of time. The developed technique for solving Heyawake puzzles uses swarm intelligence (based on the work done by Dorigo et al. [1]) because the scents used by the ants for communication can be related to the rules of the Heyawake puzzle.

It is expected that the developed techniques can be applied to other problems. One example is course scheduling at a university. The courses must fit within specified time slots, only one class may be scheduled in a room at a time, the classes for one major must not be scheduled at the same time, etc. All related factors must be considered to find the optimal schedule of courses. Both course scheduling and Heyawake are similar because of multiple constraints.

The next two sections discuss the Heyawake puzzle and its rules.

[1] Japanese, meaning "divide a room"

1.1 Heyawake Puzzle

The puzzle consists of a grid of $m \times n$ cells. The cells are divided into rooms of varying sizes and are indicated by a bold line. An optional room restriction is designated by a number in the room. Figure 1.1 shows a sample puzzle.

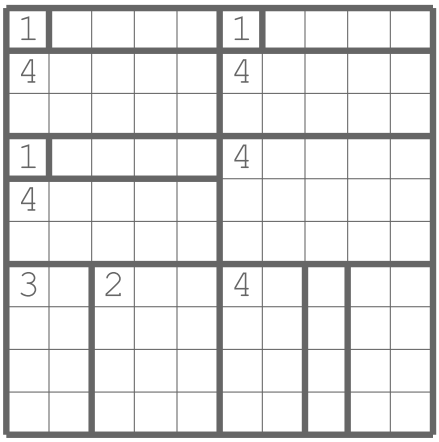

Figure 1.1: Sample Heyawake Puzzle [4]

1.2 Heyawake Rules

The objective of the game is to fill the cells according to four rules [3]:

1. A room containing a number must have that exact amount of filled cells. A room without a number can have any amount of filled cells.
2. A single line of white cells may not exceed two rooms in length.
3. Filled cells cannot touch, except diagonally.
4. All white cells must be connected in a single path. There must be no isolated blocks of white cells.

Figure 1.2 illustrates the violation of each rule and Figure 1.3 shows the solution.

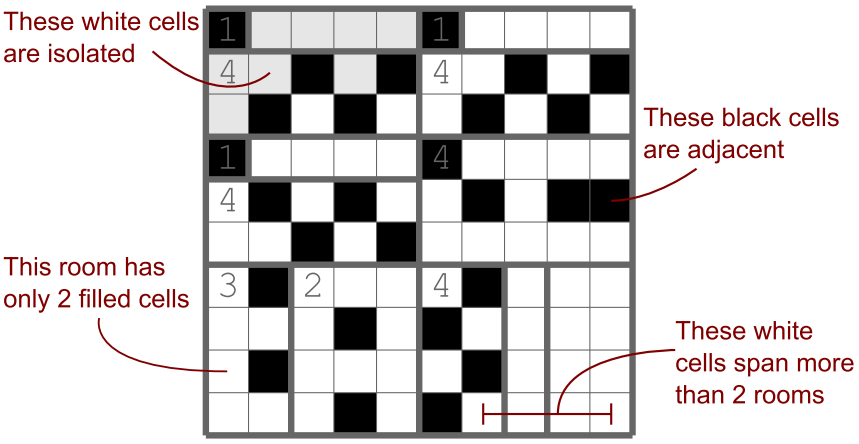

Figure 1.2: Rules Illustration

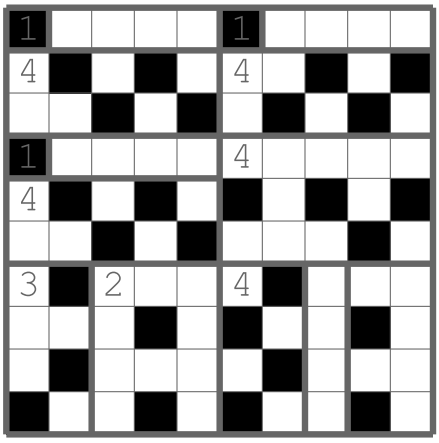

Figure 1.3: Sample Heyawake Solution [4]

Chapter 2

Method for Solving Heyawake

The developed approach for solving the Heyawake puzzle uses a software learning technique that is based on the ant colony optimization techniques developed by Dorigo et al. [1]. Software ants perform simple tasks by placing and following scents. The ants leave scent to communicate with other ants when something of interest or a hazard is found. The artificial scent is similar to the scent left behind by real ants when they find food or an obstacle. In the heuristic, the scents correspond to the rules of the game — they indicate cells in the solution that violate the game's rules. As the ants wander through the puzzle, most follow the scent left by other ants, but sometimes they wander off on in other directions. Such swarm intelligence, provided by scents and wandering, allows the simple software ants to have learning abilities and perform complex tasks.

The developed approach to solving the puzzle is divided into two phases. Phase one considers rules one, three, and four. Additionally, rule two (span of white cells across three or more rooms) is considered in phase two. Phase one will focus within a room and phase will look across room boundaries. The two phases repeat until there are no more violations to the rules.

The adjustments made by the artificial ants will always conform to rule one. Therefore, the total amount of conflicts in the puzzle account for the adjacency conflicts, isolated cells, and those spans of white cells longer than two rooms. Sections 2.1 and 2.2 discuss their calculations.

For rooms without a restriction on the number of black cells, a dynamic restriction is placed on the room allowing the number to move up and down by a fraction as the puzzle is processed. The number starts at zero and is

rounded down to the nearest whole. Details of incrementing and decrementing are described in Sections 2.1 and 2.2.

2.1 Phase One

In this phase, rules one, three, and four (number of cells in room, cell adjacency, and white cell continuity) are considered. Phase one processing sends a set of n ants into the puzzle to see what adjustments each one makes to reduce the number of conflicts in the puzzle. Experiments show using 20 ants provides good performance. Each ant in the set manipulates the same base solution. The very first base solution is randomly generated. Afterwards, the solution with the least number of conflicts becomes the new basis for the next round. Phase one repeats until there are no more rule violations.

For the manipulation, each ant endeavors to reduce the number of conflicts in a randomly selected room. First, all of the cells in the selected room are cleared (set to white). Then, the cells in the room are refilled.

A room is filled using a two step process. The first step takes into account adjacency between cells (rule three). The cell with the highest scent is given a P percent chance of being filled. Providing randomness pushes some ants to follow a new path. Experiments show a P value of 70 performs well. If the cell is filled, the cells above, below, left, and to the right within the room are marked as 'locked.' The cell with the next highest scent is then given a chance to be filled, as long as it is not locked. This will continue until either the correct number of cells in the room are filled, or every cell has been given a chance to be filled. If there are still cells that need filled in the room, step two mimics the step one process ignoring the lock on the cells. By ignoring the lock, the room will be filled with the correct number of cells. Then, if the room has a dynamic restriction, the value is decreased by a small amount. The value is decreased by 0.05 so that is takes 20 fill attempts on a room before the restriction moves down one (recall that the dynamic restriction is rounded down). After both steps are complete, the puzzle is checked for adjacency and continuity conflicts.

After the room is filled, there may still be conflicts in the puzzle. The conflicts are dealt with by adjusting the scents on the cells. Adjacency conflicts can be created when the cell lock is ignored while filling the room. They can also exist between room boundaries, since filling a room does not look outside of it. To find adjacency conflicts, each filled cell is checked. For each

filled cell, if the cell above or to the left is also filled, each cell's conflict count is increased by one. The total number of adjacency conflicts is increased by two. By only looking at the cells above and to the left of a cell, and incrementing the conflict count on both cells, this algorithm reduces the number of required checks.

A correct solution has all white cells connected. To check for continuity conflicts, a disjoint–set data structure is used[1]. All white cells are scanned through and placed into sets. Connected white cells are placed into the same set. If the final result yields multiple sets, discontinuity exists. The cells that cause the isolation are the black cells that border the white areas. The isolation causing cells' conflict counts are increased by one and scents are adjusted. The largest set of cells, by count, is considered the main area; all of the white cells in the other sets are considered isolated.

After calculating the number of conflicts, scents are adjusted using a weighted moving average: $S_{new} = S_{old} \times (1 - \alpha) + S_{target} \times \alpha$, where s_{new} is the new scent, S_{old} is the old scent, S_{target} is the target scent, and α is a proportion factor. By using a weighted moving average, adjustments to the scent are smoothed and bounded. The weighted moving average also preserves history; by adjusting α, the relative importance of the new or old scent can be adjusted. In the developed method, a scent of 0 is neutral, -1 means a cell is least likely to be filled, and 1 means the cell is likely to be filled. Using a weighted moving average facilitates dissipation of the scent over time.

If a cell has a conflict count greater than zero, its scent is reduced using the weighted moving average formula (described in the previous paragraph) with $\alpha = 0.1$ and $S_{target} = -1$. By reducing the scent, the probability of the cell being filled in the future is reduced. If a cell does not have any conflicts, its scent is increased using $\alpha = 0.1$ and $S_{target} = 1$. The scent is increased because the lack of conflicts indicates that the cell is in a good position and possibly needs to remain filled in the final solution.

After adjusting the scent based on individual cell conflicts, cell scents are adjusted based on the total number of conflicts in a room. If a room does not have any conflicts (adjacency or continuity), the scent on all filled cells is increased ($\alpha = 0.1$ and $S_{target} = 1$). This is done to mark the pattern of filled cells in the room as good. Providing positive reinforcement gives future ants a higher probability of choosing this good room layout again.

[1]Weiss [5] describes the disjoint–set data structure

2.2 Phase Two

In this phase, rule number two is considered. Rule number two restricts spans of white cells to be no more than two rooms in length. First, all white cells in a large extent (spanning more than two rooms) are placed into a list. If this list is empty, the phase ends because the puzzle has been solved. Otherwise, for each of the cells in the list, the changes to the puzzle if the cell is filled are determined. The least conflicted solution is used as a basis for the next phase one execution.

To determine the effect of filling a cell in a large extent, the room the cell is in is first cleared. The cell is then filled. The remainder of the room is filled, as described in Section 2.1. The puzzle's conflicts (isolated cells, adjacency conflicts, and the number of cells in a long extent) are then found and totaled, as previously described in Section 2.1.

If the selected cell falls within an unrestricted room, the dynamic restriction value on the room is increased. In the developed method, the value is increased by 0.6 so that it takes two increases before the (rounded down) value of the restriction increases by one.

Chapter 3

Experiments

The developed technique was successfully executed on three different 10×10 cell puzzles. The puzzles used for analysis are from NIKOLI's website [4]. Puzzles one through three were solved 100 times. Table 3.1 shows the puzzle's characteristics: difficulty rating (as determined by NIKOLI), dimensions, number of restricted rooms and cells, and number of unrestricted rooms and cells. The trials were run on a computer with a 3.1 GHz AMD Athlon X2 6000+ and 6 GB of DDR2-800 SDRAM. The operating system used was Microsoft Windows 7 x64. The solver was written in Microsoft Visual C# 2008 using .NET Framework 3.5 SP1 and built for "Any CPU." Table 3.2 shows each puzzle's CPU time statistics (average, minimum, maximum, and 25th, 50th, and 75th percentiles). Figures 3.1, 3.2, and 3.3 show the trial's statistics in a box plot for puzzles one, two, and three, respectively. Each box plot illustrates the range of values as a whisker, the mean as a diamond, the median as the midpoint on the box, and the 25th and 75th percentiles as the edges of the box.

Puzzle	Difficulty Rating	Dimensions (cells)	Number of Rooms	Restricted Rooms	Unrestricted Rooms	Restricted Cells	Unrestricted Cells
1	Easy	10×10	18	9	9	56	44
2	Easy	10×10	15	10	5	76	24
3	Easy	10×10	18	12	6	70	30

Table 3.1: Puzzle Characteristics

	CPU Time (seconds)					
Puzzle	Average	Minimum	Maximum	25th Percentile	Median	75th Percentile
1	433.26251	5.397635	1481.276	163.4890	363.5135	559.2011
2	1.7819994	0.374402	8.377253	0.834605	1.201208	1.911012
3	0.5456915	0.218401	1.544409	0.343202	0.452403	0.670804

Table 3.2: CPU Time Statistics for Each Puzzle's Trials

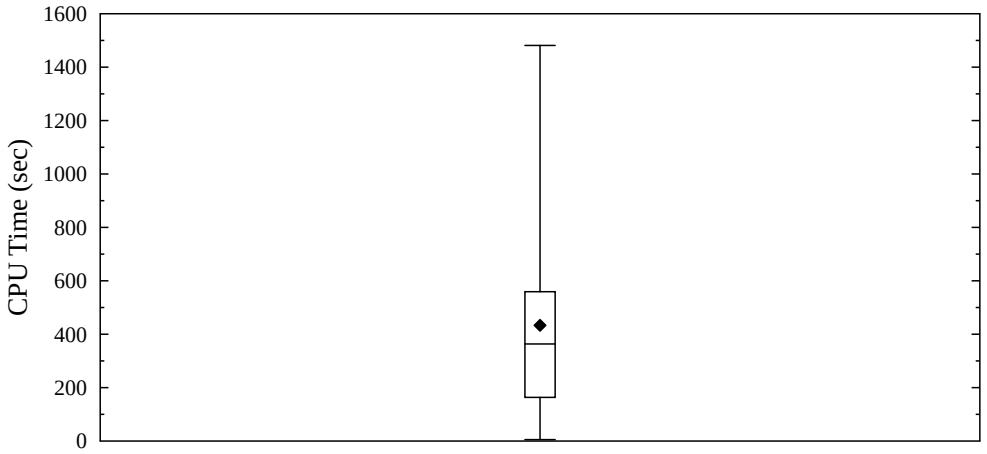

Figure 3.1: Puzzle 1 Results

The algorithm works well on puzzles with a lower percentage of unrestricted rooms (puzzles two and three). However, puzzles with a higher percentage of restricted rooms (puzzle one) take longer to solve. In all puzzles, phase one processes quickly. However, phase two starts improving the solution slowly, but speeds up once more unrestricted rooms have cells filled and the total number of rule two (long white space extents) violations decrease. Less restrictive puzzles spend more time in phase two than phase one and have many more iterations of the two phases. The changes made to the dynamic room restriction by phase two require phase one to redo much of the work already done.

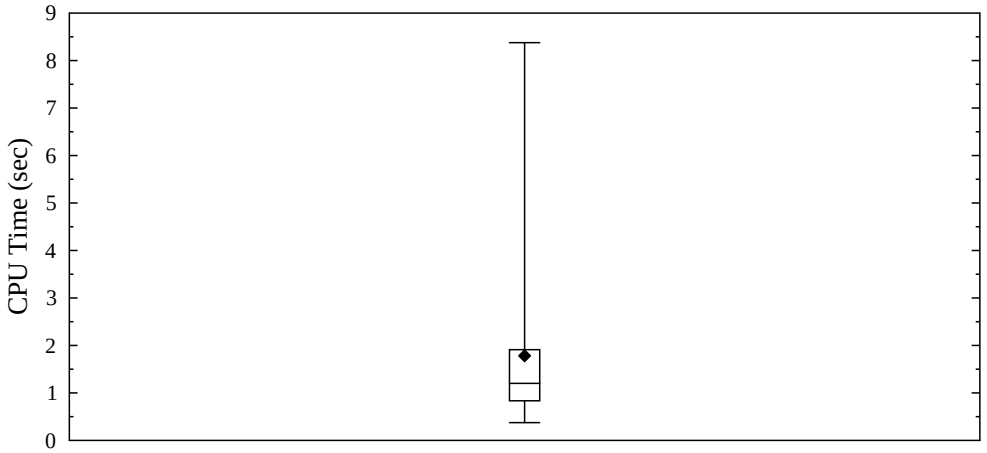

Figure 3.2: Puzzle 2 Results

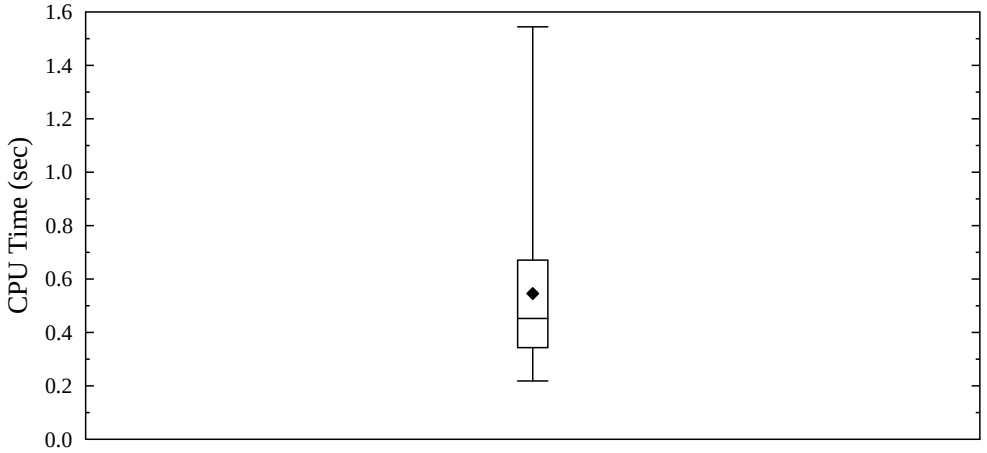

Figure 3.3: Puzzle 3 Results

Chapter 4

Conclusion

The developed swarm intelligence approach is able to solve several Heyawake puzzles. A two phase method for solving the multiple constraint puzzle was presented. The method demonstrated good performance on more restrictive puzzles (percentage of rooms with a restriction), and poorer performance on less restricted puzzles. Improvements can be made to better handle puzzles with fewer restricted rooms and to also handle larger puzzles.

Using the swarm intelligence learning technique allowed the software to experiment, learn from the results, and influence future solving attempts with the knowledge gathered. Using scents on the cells provided influence for the artificial ants to work towards a solution with fewer conflicts until the puzzle was solved (no conflicts remained).

The experiments suggest an improvement to the two phase solving method is needed because phase two invalidates the work done by phase one. Changing the restriction (dynamic) on one room causes phase one to redo some of the work already performed. Further enhancements to the method could use a single phase approach that handled all four rules simultaneously. This would reduce the amount of wasted work caused by the two phase method implemented.

An additional area to investigate in future work would be the use of multiple scents. Different scents could represent different types of violations. This could provide a way to prioritize conflict resolution in a manner that most efficiently solves the puzzle.

Bibliography

[1] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B*, 26(1):29–41, 1996. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.1865&rep=rep1&type=pdf`.

[2] Markus Holzer and Oliver Ruepp. Fun with algorithms. In *Fun with Algorithms*, volume 4475/2007 of *Lecture Notes in Computer Science*, pages 198–212. Springer Berlin / Heidelberg, 2007. doi: 10.1007/978-3-540-72914-3_18. URL `http://www.springerlink.com/content/77t44731124j6427`.

[3] NIKOLI Co., Ltd. Heyawake, 2010. URL `http://www.nikoli.co.jp/en/puzzles/heyawake/index_text.htm`.

[4] NIKOLI Co., Ltd. Sample problems of heyawake puzzle [www.nikoli.com], 2010. URL `http://www.nikoli.com/en/puzzles/heyawake/`.

[5] Mark Allen Weiss. *Data Structures and Algorithm Analysis in C++*, chapter 8, pages 317–321. Addison Wesley, 3rd edition, 2006.

Academic Vita
Gregory Kriston

gregory.kriston@ieee.org

Education

Bachelor of Science in Software Engineering, May 2010
with Honors in Software Engineering
Penn State Erie, The Behrend College, Erie, PA
Thesis: Applying Swarm Intelligence to Solve Heyawake Puzzles
Thesis Supervisor: Wen-Li Wang

Related Experience

Software Engineering Co-Op Student, June 2008 – July 2009
Northrop Grumman Electronic Systems, Baltimore, MD
- Designed and developed hardware simulation software
- Assisted in automating software build process
- Designed and developed software update mechanism
- Developed ASP.NET web based applications
- Tested and debugged software applications

IT Intern, May 2007 – Aug 2007
Bucyrus America, Houston, PA
- Provided tier one technical support

Awards

Dean's List

Presidential Scholarship for Penn State Erie, The Behrend College
2009 – 2010 and 2007 – 2008 Academic Years

Dr. and Mrs. Arthur W. Phillips Scholarship for Schreyer Scholars
2006 – 2007 Academic Year

Penn State Behrend Campus Scholarship
2005 – 2006 Academic Year

Roberts & Reider Scholarship
2005 – 2006 and 2006 – 2007 Academic Years

Memberships

IEEE, IEEE Computer Society, IEEE Communication Society

Tau Beta Nu, *Behrend Engineering Honor Society*

Phi Kappa Phi, *National Honor Society*

Activities

Erie County 4H / FIRST Robot Clubs
Mentor, January 2010 – April 2010

Penn State Behrend Robotics Club
President, August 2009 – April 2010
Treasurer, March 2007 – May 2008

Software Engineering First Year Interest Group
Mentor, August 2009 – December 2009
Mentor, August 2007 – December 2007