2101INT Assignment II: Constraint Satisfaction: Tantrix

Due Date: 10AM 18th October (Tuesday Week 12) 2005
Weighting: 35% of the total marks for the course

This assignment is to be completed **either** individually **or** as a pair. The assignment is the same in either case, but if you decide to work individually only a single report needs to be submitted.

Deliverables

Each Individual/Pair must submit:

1) A working version of your source code (written in either C/C++ or Java), and any associated makefiles or project files emailed as a ZIP file to s.bain@griffith.edu.au by the submission date. Your code must not reference/import anything apart from standard classes/libraries. You will have to demonstrate your working code on the due date from the files you submitted.

2) A printout of your source code, clearly marked with your name and student number, and instructions on how to compile and run your program.

Each Individual must also submit:

3) A report detailing:

- If (and only if) you were working as a pair, you must detail your individual contribution to the implementation aspects of the assignment (and your partner's student number)
- The features of the algorithm that you implemented
- The algorithm you implemented (in pseudo-code)
- A discussion of the performance of your algorithm. What test cases did you try? Was it able to locate optimal solutions? Did you evaluate any other algorithms that were less efficient?
- How well your algorithm scales up for problems of various grid sizes, in terms of its average case performance over a number of random runs
- Appropriate references to related work (must include references to at least one published paper relevant to the algorithm you implemented, and does not include references to the course text or lecture notes). You may find Citeseer (http://citeseer.ist.psu.edu) a useful resource to locate relevant conference and journal articles.

Students are advised that although the development of the program may be a combined effort, the reports are to be completed individually.

Assignment

Overview

Tantrix (http://www.tantrix.com/) is a game not unlike dominoes but played with hexagonal tiles. On each tile are three lines of different colours, which connect different edges of the tile. There are four possible line colours: Red, Yellow, Green and Blue. There are three different types of lines: acute, obtuse and straight. One of the objectives of Tantrix is to arrange a set of tiles to form a loop with as many segments as there are tiles, that is, the loop passes through every tile. Two samples are shown in Figure 1.

Figure 1: (a) A red-5-loop and (b) a blue-6-loop

The complete set of 56 Tantrix tiles is shown in Figure 2.

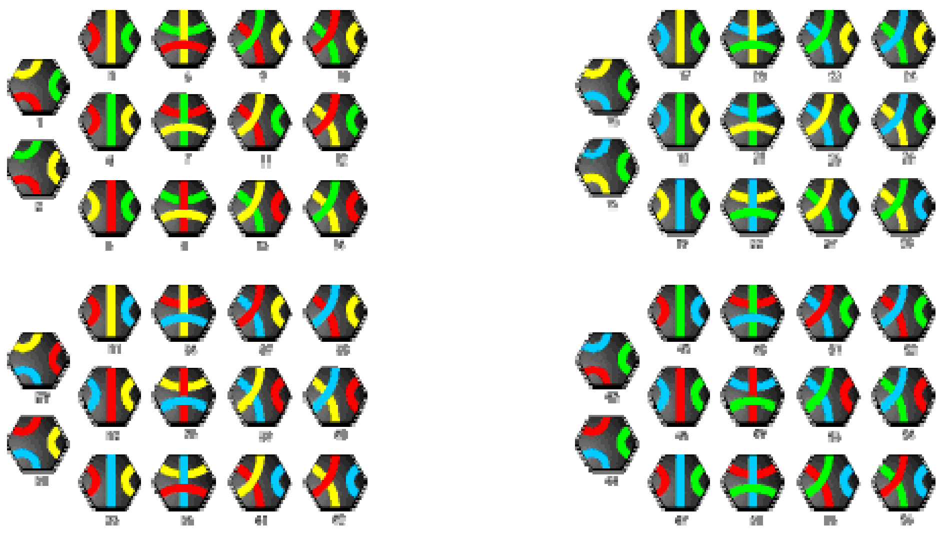

Figure 2: The 56 different Tantrix tiles

The purpose of this assignment is to write a program that can solve Tantrix loop puzzles for an arbitrary sized x-by-y grid.

Instructions

Solving a Tantrix loop puzzle on an x-by-y grid is easily viewed as a constraint satisfaction problem. The variables are the positions on the grid, as shown below.

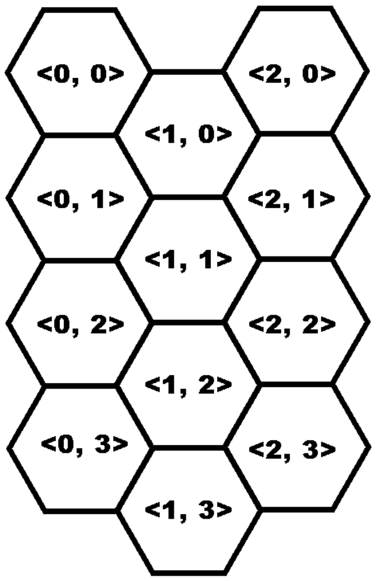

Figure 3: An x-by-y grid of hexagonal tiles.

This is a 3-by-4 grid, so has 3 columns of 4 tiles.

Tiles are described by their column and row position as <x, y>, just as squares in a rectangular grid might be.

So an x-by-y grid gives rise to x*y CSP variables.

The values that may be assigned to these variables are the available tiles, but since each tile may be placed in any of 6 different orientations, the number of possible values is 6 times the number of available tiles. Your program will accept as input from the user the list of tiles it is to use to solve the problem.

There are a number of *mandatory* constraints in this problem:

- Each tile must be placed in only one position, with one orientation
- Each location must have exactly one tile
- The two edges of adjoining tiles must be of the same colour

There is one additional *soft* constraint:

- Your program must create the largest possible loop of a user specified colour

Implementation

Your program must be a *local search* procedure that can locate solutions to the problem specified above. You must implement something more than a greedy local search procedure if you want to obtain full marks. This can be an algorithm like tabu search; weighting; another algorithm from the literature; an algorithm using "undefined" values (see: www.ijcai.org/papers/0798.pdf); or even a particularly novel cost function that improves the search.

Your algorithm will be assessed on the quality of solutions found and its efficiency at locating solutions.

A sample implementation containing many of the data structures and functions you will need (but only a random guess heuristic) is available on the subject homepage.