Constraint Technology (course 1DL023)
Autumn 2008 – Project Portfolio

Pierre Flener

Department of Information Technology, Uppsala University, Sweden

— Hard deadline: 18:00 on Friday 24 October 2008 —

Submission Instructions

1. Pick *one* project from this portfolio.
2. The project is to be done *individually*. Take our warning on plagiarism very seriously. *We assume that by submitting a solution you are certifying that it is solely your work.*
3. Specify the operating system, CPU, CPU speed (in GHz), and cache size (in KB) of the computer you chose for this project.
4. Document your program by:
   - Instructions on how to compile and run it.
   - Sample test-run commands with inputs and outputs. (Check whether these test runs are reproducible by the program you submit.)

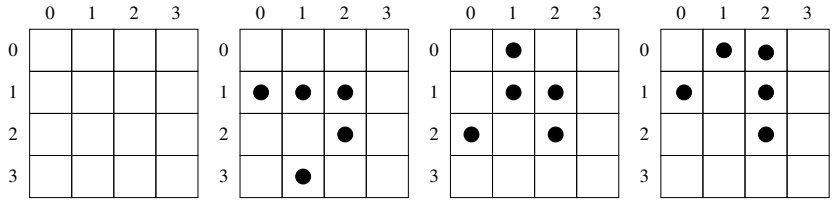
   If your program has this documentation, compiles without error under version 2.1.2 or 2.2.0 of Gecode/J, and produces correct outputs on our computer, then you get 6 points, otherwise you get 0 (zero) points.
5. Give a mathematical description of your model (in pseudocode, as in the lectures), namely its decision variables, redundant decision variables, domains of the decision variables, instance variables, problem constraints, channelling constraints, implied constraints, symmetry-breaking techniques, branching heuristics, search order, etc. (8 points)
6. Experimentally validate the positive impact of any redundant decision variables, channelling constraints, implied constraints, and symmetry-breaking techniques. (3 points)
7. Experiment with at least eight combinations of (not necessarily predefined) variable and value ordering heuristics. Identify the seemingly fastest combination until the *first* optimal solution and explain why it beats the other ones. For each combination, give the CPU runtime on the chosen computer and the number of failures until the *first* optimal solution, in tabular or plotted form. (3 points)
8. Discuss your results. Justify all claims. State any assumptions you make. Write clear code and documentation. Spell-check and grammar check your report.
9. Put your answer files into a folder named *Lastname.Firstname*. Do not use any special characters in the folder name. Include only source code files, that is remove any executable files. Textual documentation must be in *.pdf* or *.txt* format. Include a *ReadMe.txt* file to explain the purpose of each other file. Make a *.tgz* or *.zip* compressed archive of maximum 1MB from this folder. *Verify whether it decompresses properly, reproducing your folder exactly, and actually corresponds to your chosen project.*

10. Submit your compressed folder via the Course Manager server (whose clock may differ from yours) by the deadline given above. This deadline is hard and no other method of submission will be accepted, except in cases of force majeure. Late solutions will be penalised by 1 point for each 12h of delay, but solutions that are late by more than 96h will get 0 (zero) points.

Failure to follow the instructions above may result in 0 (zero) points, as we reserve the right to process your solutions mechanically.

Option 1: Maximum Density Still Life (20 points)

The following description is inspired by [1]. The *Game of Life* was invented by John H. Conway in the 1960s (see `http://mathworld.wolfram.com/Life.html`). The game is played on an $n \times n$ board. On this board, each cell is either *dead* or *alive*. A *state* of the game at some time is a board where some of the cells are alive and the rest of the cells are dead. Here are four states of the game for $n = 4$ where dead cells are empty and alive cells contain a black checker:



Note that each cell on the board may be dead, as in state (a). We assume that the board is extended infinitely by dead cells in all directions. Then each cell C has eight neighbours:

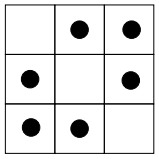
N1	N2	N3
N8	C	N4
N7	N6	N5

Between two times t and $t + 1$, the state may change under the following rules:

- If a cell has exactly three living neighbours at time t, then it is alive at time $t + 1$.
- If a cell has exactly two living neighbours at time t, then it is in the same state at time $t + 1$ as it was at time t.
- In all other cases, the cell is dead at time $t + 1$.

States (b) to (d) above are three consecutive generations. In state (b), the dead cell (1, 0) has three living neighbours, hence it is alive in state (c).[1] Similarly, the dead cell (0, 2) also has three living neighbours and becomes alive in state (c). The alive cell (0, 1) in state (b) only has one living neighbour, hence it is dead in state (c). The same thing holds for the alive cell (1, 3).

A *still life* is a state at some time t that does not change at time $t + 1$. For example:



Another example of a still life is state (a) above. A *maximum density still life* is a still life on an $n \times n$ board where the number of alive cells is maximised. For example, the still life above is also an example of a maximum density still life for $n = 3$. Indeed, there is no other 3×3 still life where the number of alive cells is greater than six. The maximum densities of still lives on $n \times n$ boards for $n = 3, \ldots, 13$ are $6, 8, 16, 18, 28, 36, 43, 54, 64, 76, 90$.

Write a constraint program for finding and displaying *all* maximum density still lives for $n > 0$. Your program must find its *first* optimal solution for $n = 7$ within three CPU minutes on your chosen computer. Report the highest value n_{max} of n where it finds its *first* optimal solution within one CPU hour on your chosen computer. You will be exempted from the exam with an A grade (ECTS) if your program solves a currently open instance ($n \geq 14$, as far as we know) in one CPU day on our computer.

[1] We use the convention (column, row) and number from zero from the upper left corner, so (1, 0) means the cell in the second column and first row.

Option 2: The Scene Allocation Problem (20 points)

The *Scene Allocation Problem* [2] is an optimisation problem where the task is to minimise the total actor cost when shooting scenes for a movie. It is described as follows in [2]:

> *The scene-allocation problem* [...] *consists of deciding when to shoot scenes for a movie. Each scene involves a number of actors and at most 5 scenes a day can be filmed. All actors of a scene must, of course, be present on the day the scene is shot. The actors have fees representing the amount to be paid per day they spent in the studio. The goal of the application is to minimize the production costs.*

The instance that we consider in this project is given as follows. The actors and their daily fees are as follows:

Actor	De Niro	Kidman	Cruise	Thurman	Liu	Pacino
Daily Fee	26,481	25,043	30,310	4,085	7,562	9,381
Actor	Stormare	Johansson	Cole	Williams	Bogart	
Daily Fee	8,770	5,788	7,423	3,303	9,593	

The nineteen scenes to shoot over at most five days and their actor sets are as follows:

Scene	**Set of Actors**
1	{Pacino}
2	{De Niro, Pacino, Liu, Thurman}
3	{Johansson, Cruise, Cole, Liu}
4	{Kidman, Cole}
5	{Cole, Stormare, De Niro, Johansson, Williams}
6	{Bogart, Johansson, Stormare, Cruise, Williams}
7	{Kidman, De Niro}
8	{Cole, Thurman}
9	{Kidman, Johansson, Cole, Cruise, Bogart}
10	{Kidman, Johansson, Cruise, De Niro}
11	{De Niro}
12	{Pacino, Bogart, Johansson, Thurman, Liu}
13	{Pacino, Thurman, Kidman, De Niro}
14	{Stormare, Cruise}
15	{Bogart, Thurman, Johansson, De Niro}
16	{Cruise, Johansson, Kidman, Cole}
17	{Cruise, De Niro, Liu}
18	{Cruise, Johansson, Pacino, Bogart}
19	{Kidman}

A non-optimal solution (of cost 581,736) to this instance is given below in two different but equivalent ways: Table (a) shows the set of scenes to shoot on a particular day, while Table (b) shows the day to shoot a particular scene.

(a)

Day	**Set of Scenes**
1	{1, 2, 3, 4, 5}
2	{6, 7, 8, 9}
3	{10, 11, 12}
4	{13, 14, 15, 16, 17}
5	{18, 19}

(b)

Scene	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Day	1	1	1	1	1	2	2	2	2	3	3	3	4	4	4	4	4	5	5

Write a constraint program for finding and displaying *all* optimal solutions to *any* instance of the scene allocation problem. Report the CPU runtime on the chosen computer and the number of failures that your program takes to find its *first* optimal solution (of cost 334,144) to the given problem instance.

References

[1] Barbara M. Smith. Maximum density still life. Problem 32 at http://www.csplib.org/.

[2] Pascal Van Hentenryck. Constraint and integer programming in OPL. *INFORMS Journal on Computing*, 14(4):345–372, 2002. Available at http://ormstomorrow.informs.org/archive/spring03/InformsArticles/constraint%20and%20IP%20in%20OPL.pdf.