Modelling for Combinatorial Optimisation (1DL449) Autumn 2015
Project: The Parking Lot Layout Problem

Max Block

January 7, 2016

Contents

1 Introduction

The original problem definition is as follows:

> In the city of Car Springs, there is a parking lot with room for 7 × 7 cars. The only entrance/exit is through the gate, and the rest of the parking lot is enclosed by a fence.
>
> The valet wants to fit as many cars as possible into the parking lot. However, it should be possible for every parked car to reach the exit without having to move any other cars.

Figure 1 depicts an example feasible (sub-optimal) solution for the case 7×7, with the cars in blue, and paths in dashed grey.

The puzzle is from a mathematical competition for Russian upper schoolers called Математический праздник — Matematicheskij Prazdnik — which can be roughly translated to 'Celebration of Maths'. The problem constructors did manage to fit 28 cars, but did not prove that their solution was optimal.[1]

This report proves an optimal solution to this instance, and discusses the general case of rectangle shaped parking lots of size $m \times n$.

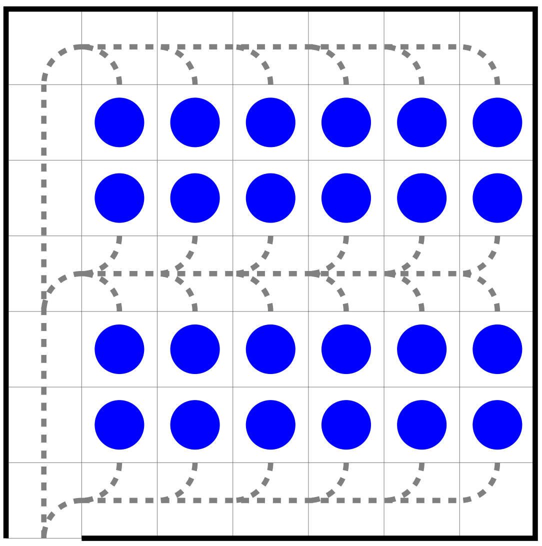

Figure 1: Example parking of 24 cars in the 7×7 parking lot.

[1]Thanks to Valentina Chapovalova for digging up the problem! Her blog entry about the problem can be found at `http://mattebloggen.com/2009/02/mattegata/`

2 Problem description

2.1 Notation

The parking lot is divided into $m \cdot n$ *cells*. Each cell may be either part of the *path*, or a *car*. A cell which is part of the path may also be called *a path (cell)*, but enough distinction will be made to avoid confusion.

A configuration of cars and path cells which uses the maximum number of cars for a pair (m, n) is called *optimal*. Two cells sharing a border in the figure is called *adjacent*. Note that diagonal cells are **not** adjacent.

The cell corresponding to the entrance may be called the *entrance* or the *exit*.

2.2 Problem constraints

- A cell is either a car or a path cell.
- Each car must be adjacent to a path cell.
- Each part of the path must be either the entrance, or connected to a cell fulfilling this constraint.[2]
- One of the corners is part of the path.

2.3 Justification of the first constraint

The first constraint is not obvious. A parking lot where a number of cells form a closed figure (e.g. a rectangle) should be a valid parking. However, this configuration does not fulfil all problem constraints. since the cars in the enclosure are not adjacent to a path.

As part of symmetry breaking I am removing these cases. To justify this, one can imagine the closed figure is opened by moving a car on the edge to an empty cell in the enclosure. If the configuration was valid, this is possible.

The same reasoning is applicable to the corners, where the same situation may occur if the very corner is empty, but enclosed by its two adjacent cells.

3 Instance description

Instances are easy to come by since the only constraint on m, n is that they are positive integers.

[2]In plain English, the dashed grey line must lead to the entrance.

4 Modelling process description

1. Solving the problem on paper to understand how I actually solve the problem, and what the computer can help with.
2. Created a frame; a parking lot (2-dimensional array of booleans[3]) and output of the parking lot (0 for a car, `.` if empty) into a rectangle in the console.
3. Added variable keeping track of number of parked cars, and redundant variables for simplifying modelling:
   - `path` is the negation of the parking lot; `true` iff the cell is part of the path. This eliminated the need of `not` in a lot of places. A channeling constraint is used to make sure this actually is the negation of `lot`.
   - `dist` is the distance to the entrance for each cell.
4. Found constraints modelling the problem constraints.
   - The third problem constraint was hardest to model. The global constraint for detecting paths (`path_from_to`) felt too general, as it considered general digraphs. My problem is an undirected graph isomorphic to some lattice-like graph.
   - Instead i decided to keep track of lengths to the exit for every cell in the path.
5. Found that the number of cars has an easy-to-compute upper bound, making proof-of-optimality quicker.
6. Broke corner symmetry.
7. Found that one can park along the walls in a predefined way.

5 Model description

5.1 MiniZinc implementation of the problem constraints

The first constraint is modelled by a 2-dimensional array (`lot`) of booleans, storing which cells are inhabited by a parked car. In order to minimize the number `not` keywords in the source code, another array (`path`) where each entry was the logical negation of the corresponding in `lot` was created.

The second constraint is propagated by a conjunction of four clauses claiming that there is a path in either direction.

To model the need for the path to be cohesive, an array `dist` of the same size as `lot` built up of integers was made. The integer corresponds to the distance to the exit, and the exit has distance 0. Each path cell has distance exactly 1 more than some adjacent cell. To make sure the path does not go through a parked car, the distance on a car cell is $m \cdot n$.

[3] And made the reasoning in 2.3.

5.2 Upper bound on number of cars

The search space is reduced by the following theorem:

Theorem 1. *For* $\min(m,n) > 2$*, the number of cars can never be higher than* $\frac{2\cdot m\cdot n}{3}$.

Proof. Apart from the first and last cells in the path, each of these cells is adjacent to two other path cells and *at most* two cars.

The first cell is adjacent to at most one car in an optimal parking; had it been adjacent to two cells the path would have been blocked and the parking could not be optimal since $\min(m,n) > 2$.

The last cell can be adjacent to three cars; one on either side, and one in front.

If the length of the path is p, then the number of cars parked along the mid section of the path is at most $2\cdot(p-2)$.[4] Furthermore, the first cell is adjacent to at most 1 car, and the last to at most 3. Hence, the number of cars adjacent to the path (and thus able to fulfil the problem constraints) is at most $2p-4+1+3=2p$. The number of cells being either a path or a car is $3p$. Since there are $m\cdot n$ cells in total, $3p \le m\cdot n \iff p \le \dfrac{m\cdot n}{3}$. The cells not part of either can be disregarded by 2.3. From this inequality, the proposed inequality is easily deduced. □

5.3 Symmetry

5.3.1 Reflectional symmetry in the diagonal

If both dimensions are larger than 2, both of the two cells adjacent to the entrance cannot be cars in an optimal solution. We may without risking solutions select which of these two is a car.

5.3.2 Corner symmetry

Not both cells adjacent to the entrance can be cars in an optimal solution, and either one can be chosen.

5.3.3 Parking along walls

We may choose 2 walls which have cars parked along the whole wall (apart from the entrance). The proof is divided into a couple of cases, and each case is easy to prove.

For visualisation, please see to figure 1. For this configuration, we may as well move the bottom row of parked cars one step down. I claim that in *every* valid configuration, it is possible to make sure the bottom border has cars parked along it. Furthermore the left wall can be filled similarily from the third cell up, to make room for the bottom row.

The proof is left as an exercise.

[4] If the path is not straight, some of these cars could have been counted twice making the upper bound lower.

6 Search description

6.1 Backends

I have compared the following backends:

1. *Gecode*
2. *Opturion CPX*
3. *Mistral*
4. *Chuffed*

Due to the problem handling mainly booleans, one could expect that the backends using SAT techniques would perform well.

6.2 Search annotations

I propose two different search heuristics:

1. Choosing variable by ability to block up the path. Hence, variables should be chosen by increasing manhattan distance to the entrance. Best-first implies letting the cell being empty.
2. Letting the backend decide by omitting search annotations.

6.3 Machine specification

The resulting model(s) was run on `trygger` in the Uppsala University computer lab. The machine is running Ubuntu 14.04.3 on 16 processors at 1.6 GHz using 8 MB cache and 24 GB memory. Since some of the back ends did not play nicely with multiple processes, all results are using only one of these cores.

6.4 Tested instances

In order to make reading the test results — and testing — somewhat managable, lines are plotted rather than surfaces. Therefore, only quadratic instances were tested. In my own experience, the relative performance for the backends are the same as in the general rectangular case.

7 Experimental results

The runtimes for the quadratic instances with side lengths 4..9 can be seen in figure 2, and the geometric means of the runtimes for the different models are seen in table 1.

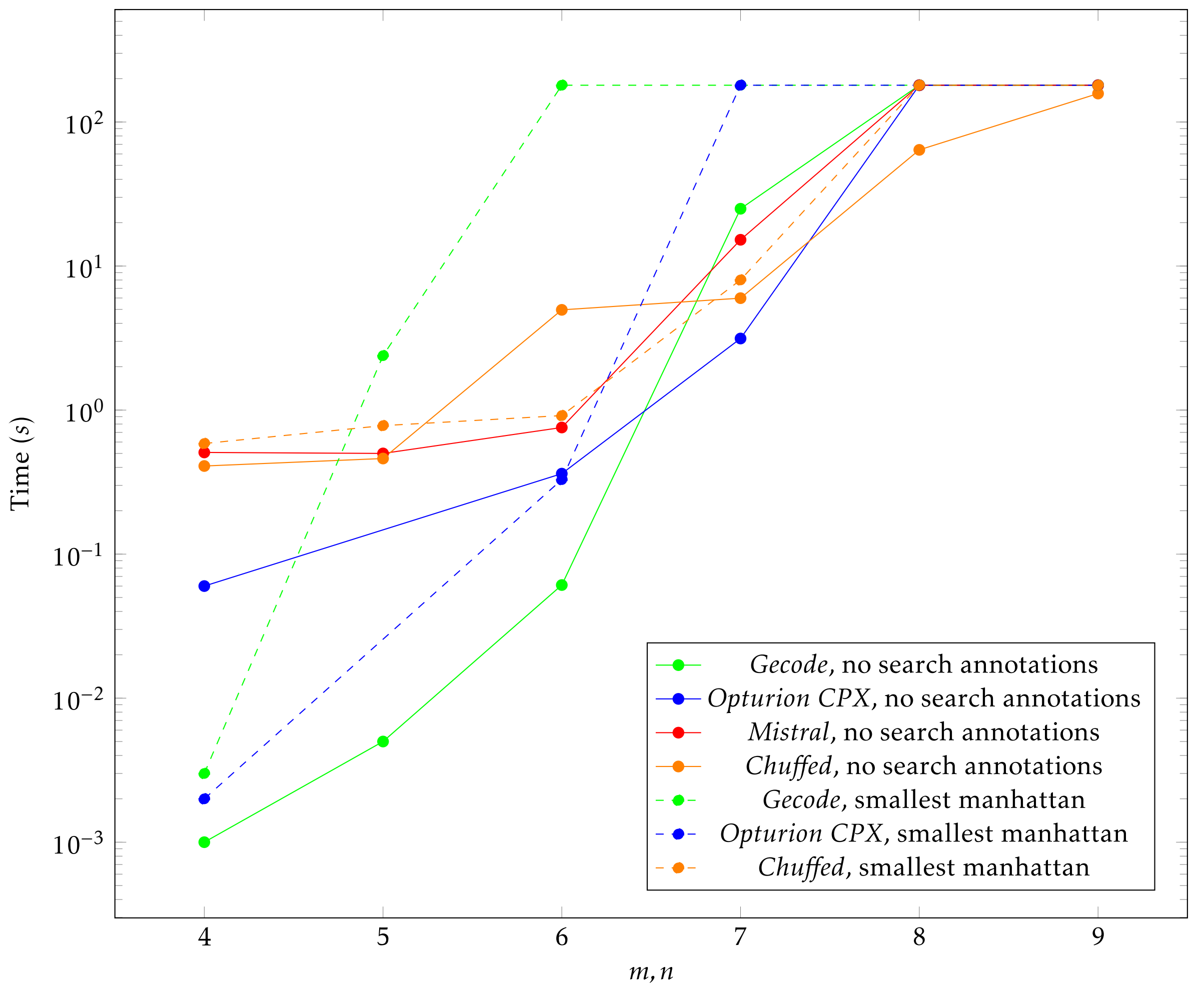

Figure 2: Runtimes until proved optimality for instances $m = n; m = 4..9$.

Back end	Search	Mean
Gecode	Default	> 0.79206
Opturion CPX	Default	> 4.66516
Opturion CPX	Manhattan	> 5.20965
Chuffed	Default	$= 6.19545$
Mistral	Default	> 6.75254
Chuffed	Manhattan	> 6.90773
Gecode	Manhattan	> 13.99622

Table 1: Geometric means of the runtimes.

8 Conclusion

8.1 My model of choice

Gecode with no search annotations has the smallest (lower bound) geometric mean of runtimes. To bring this bound to that of its closest competitor, the backend would have to run the model for extremely long. Even though giving *Gecode* all 16 cores for five minutes yielded no solution.

However, although *Gecode* solved the smallest instances very quickly, it has a much steeper curve than *Chuffed* running either model. *Chuffed* running the model version with no search annotations managed to solve all instances within three minutes

8.2 Optimality of the example instance

It turns out the maximum number of cars able to park in the 7×7 instance actually is 28. *Opturion CPX* without search annotations managed to solve this instance in 3.14 seconds.

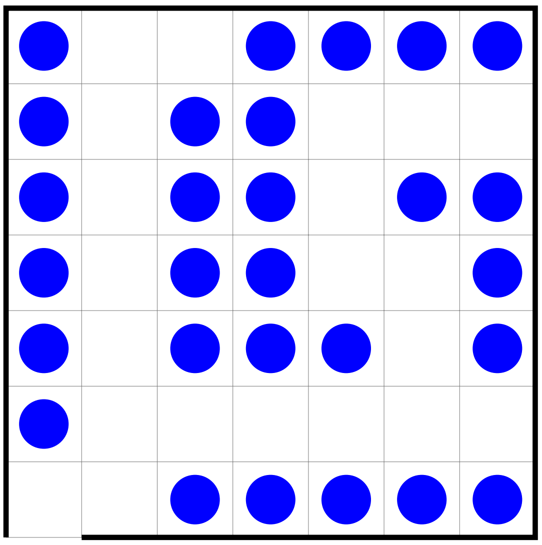

Figure 3: Example optimal parking of 28 cars in the 7×7 parking lot.

9 Reflections

9.1 Impact of implied constraints/symmetry breaking

The runtimes for the *'core'* model without no intelligent reasoning behind have not been compared to the final model due to me not wanting the report to be too long. However, it is quite amazing what difference these constraints do to the runtime; the best time among the backends I tried after making the core model for the 7×7 instance was around 50 minutes. This was brought down to little over 3 seconds, which is a speedup of 1000.

9.2 Strengthen upper bound

Although I cannot prove it, I believe the upper bound can be somewhat tightened; Each time the path turns or divides, a car is 'counted' twice. By finding a closed formula for the minimum number of turns in an optimal solution, the upper bound should be tightened. The proof-of-optimality is the largest part of solving the problem using my model, and three fourths of the time solving the 7×7 parking lot is after the optimal solution has been found.

9.3 Final thoughts

Chuffed found an (optimal) solution to the 9×9 parking lot:

```
..0000000
0........
0.0000000
0.0000000
0........
0.0000000
0.0000000
0........
0.0000000
ncars: 50
```

Where '`.`' represents an empty cell, and '0' has a car parked in it, and the upper left corner acts as the entrance. Generally, this simple striped pattern is what emerges as solutions. Initially, I had hoped for spirals or tree-like structures. However, one can easily show that the stiped pattern will approach the optimum of $\frac{2}{3}$ filling as m, n grows.

Meta-reflection

I think selecting a good subject was hard, and in hindsight I am not sure I would have chosen the same project again. This because I am not sure this problem was the best way to show what I have learned.

The thing I like the best is the speedup/search space reduction when finding nice symmetry breaking or implied constraints.

The project instruction PDF was well formulated which helped planning and writing the report. Jean-Noël was very helpful – and swift to reply – over email which of course helped a lot when stuck.

For future course instances, maybe a list of example projects would be nice. This would have made finding a suitable level of difficulty easier.

Publication

I consent to publication of this report and any file/comment published to the Studentportalen file area, including `parking.mzn`, as long as usage is properly referenced.