The Room Walk:
a constraint optimization problem

federico.serafini@studenti.unipr.it

30/08/2022

Contents

1 The problem

Let us consider a $n \times n$ shaped room, with access door I located at cell $(1, 1)$ and exit door O located at cell (n, n) (see picture, with $n = 6$). The room can contain $1 \times k$ or $k \times 1$ walls. The allowed moves are either horizontal of vertical. The problem is to identify a (the) minimal length path from I to O. As tie breaking for length, less turns in the path are preferred. If necessary, you can solve the corresponding decision problem based on path length.

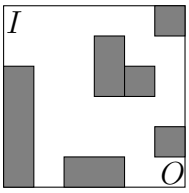

Figure 1: 6×6 room

2 Modeling

A matrix is the best-suitable data structure for room and path representation. Instead of using a single and more complex matrix to represent both of them, two different but simpler matrix are used and the relations between them are built using constraints.

2.1 Parameters

- Room size $n \in \mathbb{N}$.
- Room R is a $n \times n$ binary matrix, where $\forall i, j \in [1 \,..\, n]$:

$$R[i, j] = \begin{cases} 1, & \text{if there is a wall;} \\ 0, & \text{otherwise.} \end{cases}$$

  R is randomly generated using a C program and all of its values are set and defined (no decision variables in R, it is an input parameter of our model).

2.2 Variables

- Path P is a $n+2 \times n+2$ binary matrix, where $\forall i, j \in [1 \,..\, n+2]$:

$$P[i,j] = \begin{cases} 1, & \text{if it is a step of the path;} \\ 0, & \text{otherwise.} \end{cases}$$

  P contains the decision variables of our model and its increased size wrt R is due to the zero-padding constraints, needed to simplify a lot the neighborhood exploration (see later).

2.3 Constraints

2.3.1 Static constraints

In this document, a constraint is called *static* if all the information needed for its evaluation are available at "compile time".

- Zero padding (first row, first column, last row, last column):

$$\begin{aligned} &\forall j \in [1 \,..\, n+2] : P[1,j] = 0 \\ &\wedge \\ &\forall i \in [1 \,..\, n+2] : P[i,1] = 0 \\ &\wedge \\ &\forall j \in [1 \,..\, n+2] : P[n+2,j] = 0 \\ &\wedge \\ &\forall i \in [1 \,..\, n+2] : P[i,n+2] = 0. \end{aligned}$$

- Enter and exit cells (both must be part of the path):

$$P[2,2] = 1 \wedge P[n+1,n+1] = 1.$$

- Avoid walls (if there is a wall then the path can not step over it):

$$\forall i, j \in [1 \,..\, n] : R[i,j] > 0 \implies P[i+1,j+1] = 0.$$

Taking example room in Fig. 1 as input matrix R, after the static constraint evaluation, the matrix P looks like the following:

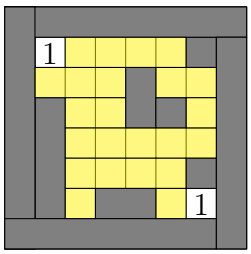

Figure 2: Matrix P after static constrain evaluation

Where:

$$P[i,j] = \begin{cases} 0, & \text{if cell is gray;} \\ 0 \vee 1, & \text{if cell is yellow (free variable);} \\ 1, & \text{otherwise.} \end{cases}$$

2.3.2 Dynamic constraints

In this document, a constraint is called *dynamic* if it introduces relations between variables whose compile-time evaluation is not possible, for example: constraints needed to connect the enter point with the exit point in order to create a path between them.

- Neighborhood for enter and exit cells (exactly one neighbor each):

$$P[2,3] + P[3,2] = 1 \wedge P[n+1,n] + P[n,n+1] = 1;$$

- Neighborhood for all the other cells in the path (exactly two neighbors each):

$$\begin{aligned} &\forall i,j \in [2\,..\,n+1] : P[i,j] > 0 \implies \\ &\Big(\neg\big((i=2 \wedge j=2) \vee (i=n+1 \wedge j=n+1)\big) \implies \\ &P[i-1,j] + P[i,j-1] + P[i+1,j] + P[i,j+1] = 2\Big). \end{aligned}$$

2.3.3 Length and turns

Thanks to the modeling decisions taken so far, the computation of length and number of turns in the path are straight forward.

- Lenght $l \in \mathbb{N}$:

$$l = \sum_{i,j=2}^{n+1} P[i,j].$$

- Turn detection is performed using a convolution operation over the non-padded section of the matrix P, with a 2×2 filter full of ones.

  The result of each filter application, is encoded in the $n-1 \times n-1$ binary matrix tc (turn counter). $\forall i,j \in [2\,..\,n]$:

$$tc[i-1, j-1] = \begin{cases} 1, & \text{if } \sum_{s,t=0}^{1} P[i+s, j+t] \geq 3; \\ 0, & \text{otherwise.} \end{cases}$$

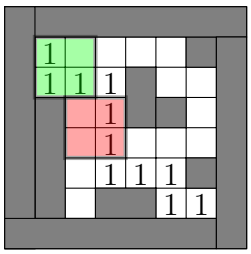

Figure 3: Turn detection in green

- Turns $t \in \mathbb{N}$:

$$t = \sum_{i,j=1}^{n-1} tc[i,j].$$

2.4 Cost function minimization

Given all the constraint discussed so far, the function f to minimize is the following:

$$f = l^2 + t. \tag{1}$$

The length squared is needed to give priority to the shortest paths and then using turns as tie breaking.

Proof. The proof is based on the fact that $\forall$ path p of length l and turns t :

$$1 \leq t < l. \tag{2}$$

Let p_1 be a path of length l_1 and t_1 turns. Let p_2 be a path of length $l_2 = l_1+1$ and $t_2 < t_1$ turns.

$$\begin{aligned}
f_1 &= l_1^2 + t_1 && [(1)] \\
&< l_1^2 + l_1 && [(2)] \\
&< l_1^2 + 2l_1 && [(2)] \\
&< l_1^2 + 2l_1 + 1 + t_2 && [(2)] \\
&= (l_1 + 1)^2 + t_2 \\
&= l_2^2 + t_2 && [l_2 = l_1 + 1] \\
&= f_2 && [(1)].
\end{aligned}$$

$f_1 < f_2$ then p_1 is preferred, no matter if p_2 has lass turns. □

2.5 Additional constraints to help the search

Constraints seen so far, together with the minimization of the cost function f, are enough to find the optimal solution. However, during the search procedure, it is possible that undesired candidate solutions are found, for example:

Figure 4: 2×2 and 3×3 squares

Solution containing this kinds of polygons will be automatically discarded since they are incrementing the cost function value but, to speed up the search procedure, constraints can be added.

- Avoid 2×2 squares:

$$\forall i, j \in [2 \,..\, n] : \sum_{s,t=0}^{1} P[i+s, j+t] \leq 3.$$

- Avoid 3×3 squares (a zero cell that is not a wall has at most two neighbors):

$$\forall i, j \in [2 \,..\, n+1] : (P[i,j] = 0 \wedge R[i-1, j-1] = 0) \Longrightarrow$$
$$P[i-1,j] + P[i,j-1] + P[i+1,j] + P[i,j+1] \leq 2.$$

- Avoid 4×4 squares (a 2×2 square of zeros without walls has at most 4 neighbors):

$$\forall i, j \in [2 \,..\, n+1] :$$
$$\sum_{s,t=0}^{1} P[i+s, j+t] + R[i+s-1, j+t-1] = 0 \Longrightarrow$$
$$P[i-1,j] + P[i-1,j+1] + P[i,j+2] + P[i+1,j+2] +$$
$$P[i+2,j] + P[i+2,j+1] + P[i,j-1] + P[i+1,j-1] \leq 4.$$

Note that there are also other polygons that can be generated and, as a future work, more constraints should be added in order to handle them:

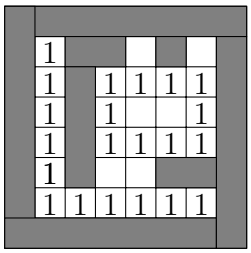

Figure 5: 3×4 rectangle

3 MiniZinc implementation

The formalization of the model discussed so far can be 1:1 translated into an effective MiniZinc program. In addition, MiniZinc allows to specify which search strategy the solver should follow, this is because for combinatorial problems some search strategies could be more effective than others. For this particular problem, finding the better strategy is not trivial: the effectiveness of a search stategy really depends on the shape of the room that is randomly generated.

3.1 Search strategy

The strategy listed below achieves good performance with a fairly large and varied sample of rooms.

- Variables: as the key of our problem, it is reasonable to start the search selecting variables from P, since the values of l, tc and t are function of it and they can be easily derived.

- Variable choice: `dom_w_deg` annotation selects the variable of P with the smallest value of:

$$\frac{|D|}{\text{w\_deg}},$$

  where $|D|$ is the domain size and w_deg is the weighted degree (the number of times the variable has been in a constraint that caused failure earlier in the search), thus allowing information gathering about the room shape from all the different explorations.

- Constraint choice: `indomain_min` annotation assigns to the selected variable its smallest domain value.

  Since $l \ll n^2$ then the number of 0 in P is greater than the number of 1; selecting the lowest value is in general the right choice (reduce backtraking).

- Restart: a bad decisions made at the top of the search tree can take an exponential amount of search to undo. Restart annotation allows to restart the search from the top thus having a chance to make different decisions every time a node limit in the search tree is reached.

With `restart_geometric(n, 1)` the k-th restart has a node limit of $1 \times n^k$. Restart and `dom_w_deg` together are found to be very effective in finding unsatisfiability.

4 Results

$\forall n \in \{ 6, 8, 10, 12, 14, 16, 18 \}$:

- 20 different room instances with $2n$ walls are randomly generated;
- 20 different room instances with n walls are randomly generated.

Results are plotted using a linear scale for the room size (x axis) and a logarithmic scale for the time (y axis). Fixing the room size and the number of walls, the time value that is plotted corresponds to the maximum time that is taken between all the 20 different instances (time in the worst case).

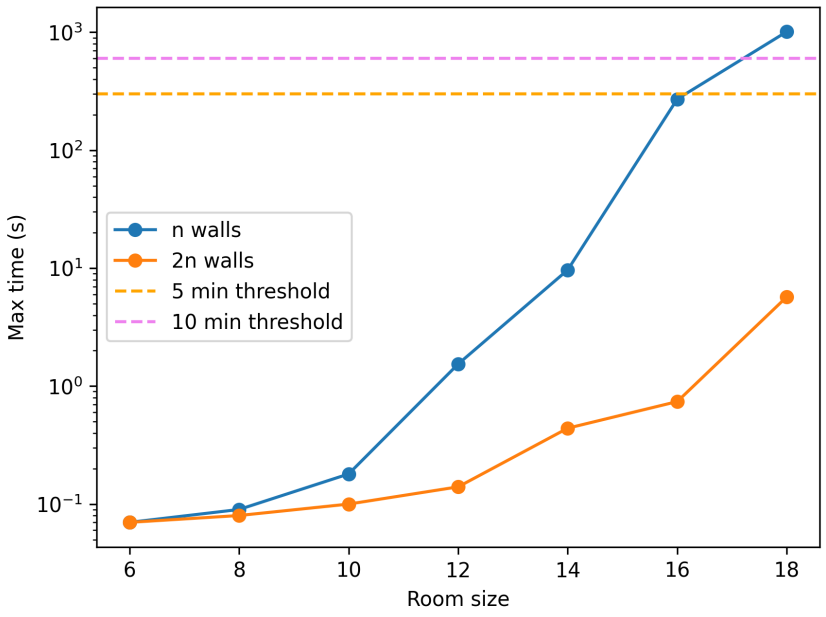

Figure 6: n and $2n$ walls

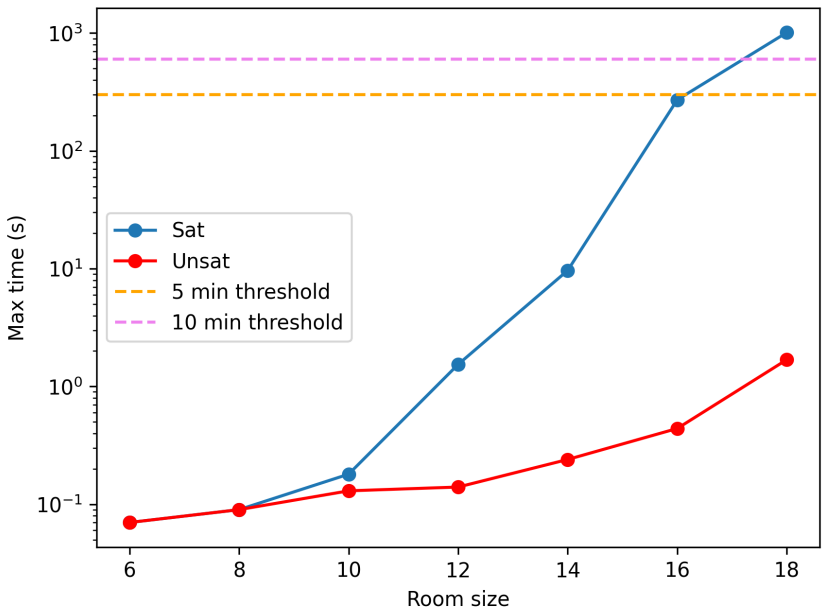

Figure 7: Sat and unsat

Not considering the padding variables, there are n^2 decision variables with a binary domain then, the search space has a size of 2^{n^2}. From the Fig. 6 it is clear that finding the solution in a room with $2n$ walls is easier, this is because the more walls there are, the fewer decision variables are involved (thanks to the "avoid walls" and "start and exit points" static constraints): let $w = |\text{wall cells}|$, then the constraint propagation, just applying node consistency, is able to reduce the search space size to 2^{n^2-w-2}.

Fig. 7 shows an appreciable feature of the model: the ability to quickly find unsatisfiability without exploring the whole search tree, thus allowing the final user to know in a short time if an admissible solution exists and then, if interested, let the solver run in order to find the optimal one. This is feasible thanks to the model design and the search strategy that allows a very effective constraint propagation.

Despite the logarithmic scale, in both Fig. 6 and Fig. 7, plotted functions seem to grow exponentially. This is due to the fact that the ratio between the wall cells and the variables, considering the noise introduced by randomization and the low number of samples, can be approximated into a slowly

descending line (Fig. 8). This means that the increasing the size, the number of variables grows faster than the number of walls, resulting in an increase of the "free variables" and an exponential impact on the search space size.

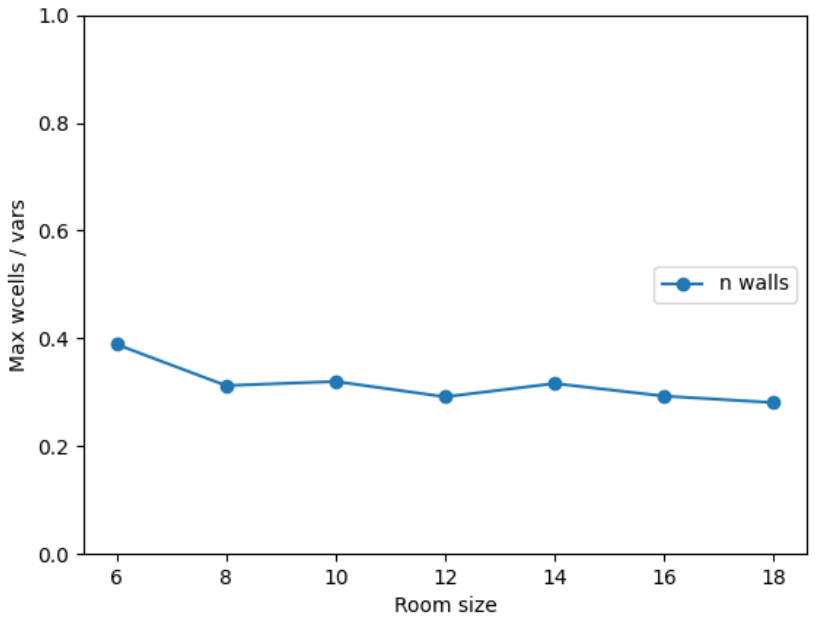

Figure 8: Ratio between wall cells and variables