UNIVERSITÁ DI UDINE

MASTER IN COMPUTER SCIENCE

ACADEMIC YEAR 2017-2018

Report of the Automated Reasoning project

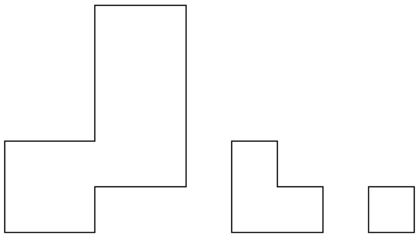
Student:
Michele Collevati

Matriculation number:
116286

E-mail:
collevati.michele@spes.uniud.it

Problem: Filling a shape

Consider a figure whose outer edges are defined by a sequence of points
(e.g., $(0, 0), (0, 2), (2, 2), (2, 5), (4, 5), (4, 1), (2, 1), (2, 0)$).

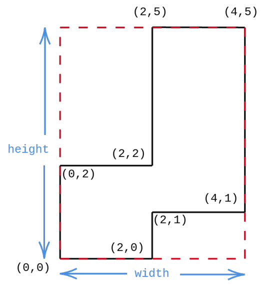


You have as many as you need "L" pieces (the biggest border is of size 2) plus unitary "U" pieces. "L" Pieces can rotate (in the same plane, they cannot be turned upside down — "U" pieces can rotate as well, but it doesn't matter too much).

The goal is to fill the internal shape of the figure, without overlapping the pieces minimizing the overall number of pieces used.

Solution

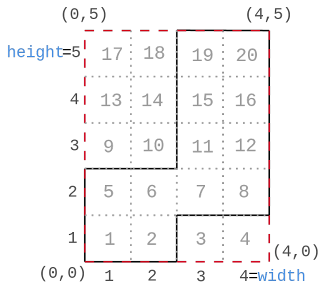
1. We obtain the minimum bounding box of the input points by calculating its width W and height H.



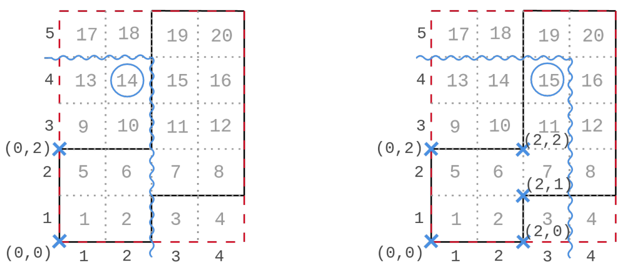
W = the maximum x coordinate of the input points
H = the maximum y coordinate of the input points

In this example, W = 4 and H = 5.

2. We think of the bounding box as if it is made up of cells. Each cell is uniquely identified by an index C, calculated from its coordinates (CX,CY) as follows: C = (CY - 1) * W + CX.



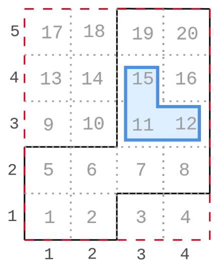
Then, we keep only the cells that belong to the shape: let C be a cell with coordinates CX and CY, and let N be the number of input points having coordinates X < CX and Y < CY. If N is even then C does *not* belong to the shape, else (N is odd) C belongs to the shape.



In this example, for the cell C = 14, N is equal to 2 and in fact it does *not* belong to the shape, whereas for the cell C = 15, N is equal to 5 and it does belong to the shape.

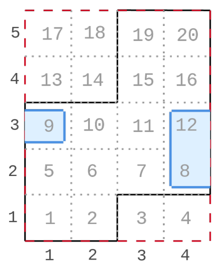
3. A cell belonging to the shape must be filled with a "L" piece or a "U" piece.

4. To form "L" pieces: the index of the cell in the middle of the piece is 1 and W respectively distant from the index of the other two cells.



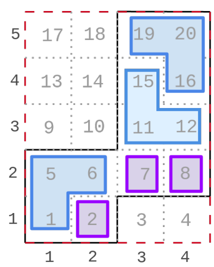
In this example, 11 is the cell in the middle of the piece and it is 1 from cell 12 and 4 from cell 15.

5. To prevent non-legal "L" pieces: a cell with multiple index of the width can *not* form a "L" piece with the next cell, because they are *not* adjacent in the bounding box.



In this example, cell 8 can *not* form a "L" piece with cell 9.

6. Objective function: minimize the number of "U" pieces.



In this example, the optimal solution is 3 "L" pieces and 3 "U" pieces.

Files

The solution includes the following files:

- `filling_a_shape.mzn`: MiniZinc encoding of the problem.
- `filling_a_shape.lp`: ASP encoding of the problem.
- `instance_generator.lp`: ASP encoding of the instance generator.
- `instance_converter.cpp`: C++ program to convert the raw instances generated by the instance generator into the .dzn and .lp format.

MiniZinc encoding

```
% Problem: Filling a shape
%
% Consider a figure whose outer edges are defined by
    a sequence of points (e.g.,
    (0,0),(0,2),(2,2),(2,5),(4,5),(4,1),(2,1),(2,0)).
% You have as many as you need "L" pieces (the
    biggest border is of size 2) plus unitary "U"
    pieces. "L" pieces can rotate (in the
% same plane, they can NOT be turned upside down -
    "U" pieces can rotate as well, but it doesn't
    matter too much).
% The goal is to fill the internal shape of the
    figure, without overlapping the pieces minimizing
    the overall number of pieces used.
%
% By Michele Collevati

% Parameters
int: num_points;  % number of points
array[1..num_points,1..2] of int: seq_points; %
    sequence of points
int: width = max([seq_points[p,1] | p in
    1..num_points]);  % width of the input points
    bounding box
int: height = max([seq_points[p,2] | p in
    1..num_points]); % height of the input points
    bounding box
int: num_cells = width * height;  % number of cells
    (area of the bounding box)
set of int: cells = 1..num_cells;
int: L_size = 3;  % size in cells of a L-piece
set of int: piece_components = 1..L_size;
int: digs = ceil(log(10.0,num_cells));  % digits for
    output

% Decision variables
array[cells,piece_components] of var 0..num_cells:
    pieces;
var 0..num_cells: num_U_pieces;   % number of "U"
    pieces
var 0..floor(num_cells div L_size): num_L_pieces;  %
    number of "L" pieces

% Constraints

```

```
% Cells that do NOT belong to the shape must be equal to 0.
% Cells that belong to the shape must be different from 0.
% Let C be a cell.
% Let N be the number of points having x and y coordinates lower than C's coordinates.
% IF N is even THEN C does NOT belong to the shape
% ELSE (N is odd) C belongs to the shape
constraint
  forall(y in 1..height,x in 1..width)
    (if (length([p | p in 1..num_points where seq_points[p,1] < x /\ seq_points[p,2] < y]) mod 2) = 0
     then forall(pc in piece_components)
            (pieces[(y-1)*width+x,pc] = 0)
     else forall(pc in piece_components)
            (pieces[(y-1)*width+x,pc] != 0)
     endif);

% Or all piece_components of a cell c are = 0 (the cell c does NOT belong to the shape)
% or all piece_components of a cell c are = c ("U" piece)
% or exactly one piece_component of a cell c is = c ("L" piece).
% To prevent a cell from having piece_components of a
% "L" piece it does NOT belong to: each cell belonging
% to a "L" piece must have a piece_component equal to
% its index.
constraint
  forall(c in cells)
    (forall(pc in piece_components) (pieces[c,pc] = 0) \/
     forall(pc in piece_components) (pieces[c,pc] = c) \/
     sum(pc in piece_components) (pieces[c,pc] = c) = 1);

% To form "L" pieces: the index of the cell in the middle of the piece
% is 1 and W respectively distant from the index of the other two cells.
% Symmetry breaker:
% pieces[c,1] is the cell in the middle of the piece
% pieces[c,2] is the cell distant 1 from the cell in the middle of the piece
% pieces[c,3] is the cell distant width from the cell in the middle of the piece
constraint
```

```
  forall(c in cells)
    (sum(pc in piece_components) (pieces[c,pc] = c) = 1 ->
      abs(pieces[c,1] - pieces[c,2]) = 1 /\ abs(pieces[c,1] - pieces[c,3]) = width);

% To prevent non-legal "L" pieces: a cell with multiple index
% of the width can NOT form a "L" piece with the next cell,
% because they are NOT adjacent in the bounding box.
constraint
  forall(c in cells)
    (if (c mod width) = 0
     then sum(pc in piece_components) (pieces[c,pc] = c) = 1 ->
               not exists(pc in piece_components) (pieces[c,pc] = c+1)
     else if (c mod width) = 1
          then sum(pc in piece_components) (pieces[c,pc] = c) = 1 ->
                    not exists(pc in piece_components) (pieces[c,pc] = c-1)
          else true
          endif
     endif);

% To tie together the 3 cells that form a "L" piece:
% they must have the same piece_components.
constraint
  forall(c in cells)
    (sum(pc in piece_components) (pieces[c,pc] = c) = 1 ->
       forall(pcI in piece_components,pcJ in piece_components)
         (pieces[c,pcI] = pieces[pieces[c,pcJ],pcI]));

% number of "U" pieces
constraint
  num_U_pieces = sum(c in cells)
                     (forall(pc in piece_components) (pieces[c,pc] = c));

% number of "L" pieces
constraint
  num_L_pieces = sum(c in cells)
                     (sum(pc in piece_components) (pieces[c,pc] = c) = 1) div L_size;

```

```
% Solve
solve :: int_search(pieces, input_order,
    indomain_max, complete) minimize num_U_pieces;
%solve :: int_search(pieces, first_fail,
    indomain_max, complete) minimize num_U_pieces;

% Output
output
[
  if x = 1 then "\n" else " " endif ++
    if fix(pieces[(height-y)*width+x,1]) = 0
    then format_justify_string(digs,"*")
    else show_int(digs,pieces[(height-y)*width+x,1])
    endif
  | y in 1..height,x in 1..width
]
++ ["\n\n"] ++
[
  "Number of \"L\" pieces = ", show(num_L_pieces),
     "\n",
  "Number of \"U\" pieces = ", show(num_U_pieces),
     "\n"
];
```

ASP encoding

```
% Problem: Filling a shape
%
% Consider a figure whose outer edges are defined by a sequence of points (e.g., (0,0),(0,2),(2,2),(2,5),(4,5),(4,1),(2,1),(2,0)).
% You have as many as you need "L" pieces (the biggest border is of size 2) plus unitary "U" pieces. "L" pieces can rotate (in the
% same plane, they can NOT be turned upside down - "U" pieces can rotate as well, but it doesn't matter too much).
% The goal is to fill the internal shape of the figure, without overlapping the pieces minimizing the overall number of pieces used.
%
% By Michele Collevati

% width of the input points bounding box
width(W) :- W = #max{ X : point(X,Y) }.

% height of the input points bounding box
height(H) :- H = #max{ Y : point(X,Y) }.

% x coordinates
coord_x(1..W) :- width(W).

% y coordinates
coord_y(1..H) :- height(H).

% Keep only the cells that belong to the shape.
% Let C be a cell.
% Let N be the number of points having x and y coordinates lower than C's coordinates.
% IF N is even THEN C does NOT belong to the shape
% ELSE (N is odd) C belongs to the shape
cell(C) :- coord_x(CX), coord_y(CY), width(W),
           PS = #count{ PX,PY : point(PX,PY), PX < CX, PY < CY },
           PS \ 2 == 1,
           C = (CY - 1) * W + CX.

% A cell filled by a "L" piece can NOT even be filled by a "U" piece.
n_u_piece(A) :- cell(A), cell(B), cell(C), l_piece(A,B,C).
```

```
n_u_piece(B) :- cell(A), cell(B), cell(C), l_piece(A,B,C).
n_u_piece(C) :- cell(A), cell(B), cell(C), l_piece(A,B,C).

% A cell filled by a "U" piece can NOT even be filled by a "L" piece.
n_l_piece(A,B,C) :- cell(A), cell(B), cell(C), u_piece(A).
n_l_piece(A,B,C) :- cell(A), cell(B), cell(C), u_piece(B).
n_l_piece(A,B,C) :- cell(A), cell(B), cell(C), u_piece(C).

% To form "L" pieces: the index of the cell in the middle of the piece
% is 1 and W respectively distant from the index of the other two cells.
% Symmetry breaker: in l_piece(A,B,C)
% A is the cell in the middle of the piece
% B is the cell distant 1 from the cell in the middle of the piece
% C is the cell distant width from the cell in the middle of the piece
0 { l_piece(A,B,C) } 1 :- cell(A), cell(B), cell(C), width(W),
                          |A - B| == 1, |A - C| == W,
                          not n_l_piece(A,B,C).

% "U" pieces
u_piece(A) :- cell(A), not n_u_piece(A).

% A cell can be filled by at most 1 "L" piece.
:- cell(C), cell(A), cell(B), cell(D), cell(E),
   l_piece(C,A,B), l_piece(C,D,E), A != D.

:- cell(C), cell(A), cell(B), cell(D), cell(E),
   l_piece(C,A,B), l_piece(C,D,E), B != E.

:- cell(C), cell(A), cell(B), cell(D), cell(E),
   l_piece(A,C,B), l_piece(D,C,E), A != D.

:- cell(C), cell(A), cell(B), cell(D), cell(E),
   l_piece(A,C,B), l_piece(D,C,E), B != E.

:- cell(C), cell(A), cell(B), cell(D), cell(E),
   l_piece(A,B,C), l_piece(D,E,C), A != D.

:- cell(C), cell(A), cell(B), cell(D), cell(E),
   l_piece(A,B,C), l_piece(D,E,C), B != E.
```

```

:- cell(C), cell(A), cell(B), cell(D), cell(E),
   l_piece(C,A,B), l_piece(D,C,E).

:- cell(C), cell(A), cell(B), cell(D), cell(E),
   l_piece(C,A,B), l_piece(D,E,C).

:- cell(C), cell(A), cell(B), cell(D), cell(E),
   l_piece(A,C,B), l_piece(D,E,C).

% To prevent non-legal "L" pieces: a cell with multiple index
% of the width can NOT form a "L" piece with the next cell,
% because they are NOT adjacent in the bounding box.
:- cell(A), cell(B), cell(C), width(W),
   l_piece(A,B,C),
   A \ W == 0,
   B == A + 1.

:- cell(A), cell(B), cell(C), width(W),
   l_piece(A,B,C),
   A \ W == 1,
   B == A - 1.

% Number of "U" pieces
num_u_pieces(NUP) :- NUP = #count{ A : cell(A), u_piece(A) }.

% Number of "L" pieces
num_l_pieces(NLP) :- NLP = #count{ A,B,C : cell(A), cell(B), cell(C), l_piece(A,B,C) }.

% Minimize the number of "U" pieces
#minimize { NUP : num_u_pieces(NUP) }.

% Maximize the number of "L" pieces
%#maximize { NLP : num_l_pieces(NLP) }.

#show l_piece/3.
#show u_piece/1.
#show num_u_pieces/1.
#show num_l_pieces/1.
```

Execution results

The benchmarks were made on the Asus N550JK laptop with an Intel Core i7-4700HQ CPU @ 2.40GHz × 8, 16GB of DDR3L 1600MHz SDRAM and Antergos Linux 64-bit operating system.

Software used:

- MiniZinc

  1. *MiniZinc to FlatZinc converter, version: 2.2.2, build: 34369965, Copyright (C) 2014-2018 Monash University, NICTA, Data61*
  2. *Gecode, version: 6.1.0, supported FlatZinc version: 1.6*

- ASP

  1. *clingo, version: 5.3.0, address model: 64-bit, libclingo version: 5.3.0, configuration: with Python 3.7.0 and with Lua 5.3.5, license: The MIT License*
  2. *gringo, version: 5.3.0, address model: 64-bit, libgringo version: 5.3.0, configuration: with Python 3.7.0 and with Lua 5.3.5, license: The MIT License*
  3. *clasp, version: 3.3.4, address model: 64-bit, libclasp version 3.3.4 (libpotassco version 1.1.0), configuration: WITH_THREADS=1, Copyright (C) Benjamin Kaufmann, license: The MIT License*

Time limit (t.l.): *5 min.*
opt. for optimal value

Search strategy used:

- on MiniZinc

  1. FF: *solve :: int_search(pieces, first_fail, indomain_max, complete) minimize num_U_pieces*
  2. IO: *solve :: int_search(pieces, input_order, indomain_max, complete) minimize num_U_pieces*

- on ASP

  1. tweety (default): *clingo --configuration=tweety*
     It uses defaults geared towards typical ASP problems[1].
  2. handy: *clingo --configuration=handy*
     It uses defaults geared towards large problems[1].

[1] More info can be found here (page 75): https://github.com/potassco/guide/releases/download/v2.2.0/guide.pdf

Instances	MiniZinc						ASP					
	FF			IO			tweety			handy		
	L pieces	U pieces	time	L pieces	U pieces	time	L pieces	U pieces	time	L pieces	U pieces	time
benchmark_instance_4-1	3 (opt.)	4 (opt.)	0.06s	3 (opt.)	4 (opt.)	0.06s	3 (opt.)	4 (opt.)	0.011s	3 (opt.)	4 (opt.)	0.014s
benchmark_instance_4-2	2 (opt.)	2 (opt.)	0.05s	2 (opt.)	2 (opt.)	0.05s	2 (opt.)	2 (opt.)	0.005s	2 (opt.)	2 (opt.)	0.006s
benchmark_instance_4-3	1 (opt.)	4 (opt.)	0.05s	1 (opt.)	4 (opt.)	0.05s	1 (opt.)	4 (opt.)	0.004s	1 (opt.)	4 (opt.)	0.005s
benchmark_instance_4-4	2 (opt.)	2 (opt.)	0.05s	2 (opt.)	2 (opt.)	0.05s	2 (opt.)	2 (opt.)	0.004s	2 (opt.)	2 (opt.)	0.005s
benchmark_instance_4-5	2 (opt.)	1 (opt.)	0.05s	2 (opt.)	1 (opt.)	0.04s	2 (opt.)	1 (opt.)	0.005s	2 (opt.)	1 (opt.)	0.004s
benchmark_instance_5-1	2 (opt.)	4 (opt.)	0.07s	2 (opt.)	4 (opt.)	0.06s	2 (opt.)	4 (opt.)	0.009s	2 (opt.)	4 (opt.)	0.006s
benchmark_instance_5-2	4 (opt.)	5 (opt.)	0.19s	4 (opt.)	5 (opt.)	0.18s	4 (opt.)	5 (opt.)	0.023s	4 (opt.)	5 (opt.)	0.028s
benchmark_instance_5-3	3 (opt.)	3 (opt.)	0.07s	3 (opt.)	3 (opt.)	0.07s	3 (opt.)	3 (opt.)	0.008s	3 (opt.)	3 (opt.)	0.009s
benchmark_instance_5-4	4 (opt.)	4 (opt.)	0.08s	4 (opt.)	4 (opt.)	0.08s	4 (opt.)	4 (opt.)	0.018s	4 (opt.)	4 (opt.)	0.028s
benchmark_instance_5-5	2 (opt.)	4 (opt.)	0.07s	2 (opt.)	4 (opt.)	0.06s	2 (opt.)	4 (opt.)	0.006s	2 (opt.)	4 (opt.)	0.006s
benchmark_instance_6-1	3 (opt.)	4 (opt.)	0.12s	3 (opt.)	4 (opt.)	0.11s	3 (opt.)	4 (opt.)	0.011s	3 (opt.)	4 (opt.)	0.014s
benchmark_instance_6-2	4 (opt.)	7 (opt.)	0.30s	4 (opt.)	7 (opt.)	0.30s	4 (opt.)	7 (opt.)	0.027s	4 (opt.)	7 (opt.)	0.028s
benchmark_instance_6-3	3 (opt.)	10 (opt.)	0.32s	3 (opt.)	10 (opt.)	0.32s	3 (opt.)	10 (opt.)	0.024s	3 (opt.)	10 (opt.)	0.027s
benchmark_instance_6-4	7 (opt.)	5 (opt.)	4.62s	7 (opt.)	5 (opt.)	4.79s	7 (opt.)	5 (opt.)	0.109s	7 (opt.)	5 (opt.)	0.084s
benchmark_instance_6-5	9 (opt.)	1 (opt.)	0.18s	9 (opt.)	1 (opt.)	0.30s	9 (opt.)	1 (opt.)	0.106s	9 (opt.)	1 (opt.)	0.111s
benchmark_instance_7-1	12 (opt.)	3 (opt.)	157.85s	12 (opt.)	3 (opt.)	64.25s	12 (opt.)	3 (opt.)	1.156s	12 (opt.)	3 (opt.)	0.370s
benchmark_instance_7-2	11 (opt.)	3 (opt.)	47.07s	11 (opt.)	3 (opt.)	12.75s	11 (opt.)	3 (opt.)	0.993s	11 (opt.)	3 (opt.)	0.271s
benchmark_instance_7-3	7 (opt.)	6 (opt.)	2.63s	7 (opt.)	6 (opt.)	2.61s	7 (opt.)	6 (opt.)	0.130s	7 (opt.)	6 (opt.)	0.093s
benchmark_instance_7-4	11 (opt.)	3 (opt.)	112.47s	11 (opt.)	3 (opt.)	81.74s	11 (opt.)	3 (opt.)	0.698s	11 (opt.)	3 (opt.)	0.266s
benchmark_instance_7-5	6 (opt.)	4 (opt.)	0.24s	6 (opt.)	4 (opt.)	0.24s	6 (opt.)	4 (opt.)	0.041s	6 (opt.)	4 (opt.)	0.045s

	MiniZinc						ASP					
	FF			IO			tweety			handy		
Instances	L pieces	U pieces	time	L pieces	U pieces	time	L pieces	U pieces	time	L pieces	U pieces	time
benchmark_instance_8-1	12 (opt.)	6 (opt.)	105.66s	12 (opt.)	6 (opt.)	167.52s	12 (opt.)	6 (opt.)	1.126s	12 (opt.)	6 (opt.)	0.476s
benchmark_instance_8-2	10 (opt.)	7 (opt.)	11.74s	10 (opt.)	7 (opt.)	11.65s	10 (opt.)	7 (opt.)	0.424s	10 (opt.)	7 (opt.)	0.282s
benchmark_instance_8-3	14	6	t.l.	14	6	t.l.	15 (opt.)	3 (opt.)	7.554s	15 (opt.)	3 (opt.)	0.873s
benchmark_instance_8-4	8 (opt.)	8 (opt.)	4.77s	8 (opt.)	8 (opt.)	4.67s	8 (opt.)	8 (opt.)	0.123s	8 (opt.)	8 (opt.)	0.138s
benchmark_instance_8-5	7 (opt.)	12 (opt.)	3.47s	7 (opt.)	12 (opt.)	3.21s	7 (opt.)	12 (opt.)	0.153s	7 (opt.)	12 (opt.)	0.149s
benchmark_instance_9-1	7 (opt.)	8 (opt.)	2.81s	7 (opt.)	8 (opt.)	2.78s	7 (opt.)	8 (opt.)	0.095s	7 (opt.)	8 (opt.)	0.103s
benchmark_instance_9-2	9 (opt.)	7 (opt.)	8.75s	9 (opt.)	7 (opt.)	8.68s	9 (opt.)	7 (opt.)	0.184s	9 (opt.)	7 (opt.)	0.188s
benchmark_instance_9-3	12 (opt.)	5 (opt.)	12.61s	12 (opt.)	5 (opt.)	12.57s	12 (opt.)	5 (opt.)	0.685s	12 (opt.)	5 (opt.)	0.369s
benchmark_instance_9-4	21	5	t.l.	21	5	t.l.	22 (opt.)	2 (opt.)	3.870s	22 (opt.)	2 (opt.)	2.958s
benchmark_instance_9-5	15 (opt.)	7 (opt.)	28.37s	15 (opt.)	7 (opt.)	10.72s	15 (opt.)	7 (opt.)	1.438s	15 (opt.)	7 (opt.)	0.937s
benchmark_instance_10-1	11	12	t.l.	11	12	t.l.	11 (opt.)	12 (opt.)	1.607s	11 (opt.)	12 (opt.)	0.543s
benchmark_instance_10-2	23	6	t.l.	23	6	t.l.	24 (opt.)	3 (opt.)	66.635s	24 (opt.)	3 (opt.)	4.298s
benchmark_instance_10-3	25	2	t.l.	25	2	t.l.	24	5	t.l.	25 (opt.)	2 (opt.)	6.555s
benchmark_instance_10-4	24	8	t.l.	24	8	t.l.	24	8	t.l.	25 (opt.)	5 (opt.)	7.648s
benchmark_instance_10-5	21	6	t.l.	21	6	t.l.	21 (opt.)	6 (opt.)	135.619s	21 (opt.)	6 (opt.)	2.849s

	MiniZinc						ASP					
	FF			IO			tweety			handy		
Instances	L pieces	U pieces	time	L pieces	U pieces	time	L pieces	U pieces	time	L pieces	U pieces	time
benchmark_instance_11-1	21	14	t.l.	21	14	t.l.	22	11	t.l.	22 (opt.)	11 (opt.)	6.477s
benchmark_instance_11-2	19	11	t.l.	19	11	t.l.	20 (opt.)	8 (opt.)	7.891s	20 (opt.)	8 (opt.)	2.339s
benchmark_instance_11-3	24	9	t.l.	24	9	t.l.	25 (opt.)	6 (opt.)	43.940s	25 (opt.)	6 (opt.)	5.379s
benchmark_instance_11-4	25	14	t.l.	25	14	t.l.	27 (opt.)	8 (opt.)	82.097s	27 (opt.)	8 (opt.)	12.680s
benchmark_instance_11-5	14	20	t.l.	14	20	t.l.	14 (opt.)	20 (opt.)	15.499s	14 (opt.)	20 (opt.)	1.451s
benchmark_instance_12-1	17	13	t.l.	17	13	t.l.	18 (opt.)	10 (opt.)	44.050s	18 (opt.)	10 (opt.)	2.620s
benchmark_instance_12-2	32	14	t.l.	32	14	t.l.	34	8	t.l.	15	65	t.l.
benchmark_instance_12-3	18 (opt.)	5 (opt.)	199.16s	18 (opt.)	5 (opt.)	185.48s	18 (opt.)	5 (opt.)	1.784s	18 (opt.)	5 (opt.)	1.434s
benchmark_instance_12-4	36	11	t.l.	36	11	t.l.	36	11	t.l.	22	53	t.l.
benchmark_instance_12-5	18	12	t.l.	18	12	t.l.	19 (opt.)	9 (opt.)	4.849s	19 (opt.)	9 (opt.)	2.334s
benchmark_instance_13-1	20	22	t.l.	20	22	t.l.	22 (opt.)	16 (opt.)	182.870s	22 (opt.)	16 (opt.)	4.238s
benchmark_instance_13-2	21	7	t.l.	21	7	t.l.	22 (opt.)	4 (opt.)	7.439s	22 (opt.)	4 (opt.)	3.324s
benchmark_instance_13-3	27	23	t.l.	27	23	t.l.	30 (opt.)	14 (opt.)	225.564s	30 (opt.)	14 (opt.)	19.180s
benchmark_instance_13-4	38	17	t.l.	38	17	t.l.	39	14	t.l.	16	83	t.l.
benchmark_instance_13-5	20	17	t.l.	20	17	t.l.	21 (opt.)	14 (opt.)	7.810s	21 (opt.)	14 (opt.)	3.304s

As can be seen from the table, better results were obtained in ASP compared to MiniZinc. In particular, very low execution times were obtained using the *handy* configuration, but for instances that have gone in time limit it has behaved very badly.