Using Global Constraints for Rectangle Packing

Helmut Simonis and Barry O'Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{h.simonis|b.osullivan}@4c.ucc.ie

Abstract. In this paper we solve the optimal rectangle packing problem using Cumulative and Disjoint2 constraints in SICStus Prolog with a novel decomposition method, together with a specialized search routine and various model enhancements. We improve the best known runtimes by up to a factor of 300.

1 Introduction

Rectangle packing is an important application domain for constraint programming, with significant research into improved constraint propagation methods being reported in the literature [1–7, 12]. In this paper we consider a particular case of rectangle packing from [9–11]. The objective is to fit all squares of size 1×1 to $N \times N$ into the rectangle with the smallest area that can contain all items. Optimal solutions are known for values of N from 1 to 25.

We introduce a simple decomposition method to define subproblems of fixed size, for which we use the standard formulation with global Disjoint2 (non-overlapping) and Cumulative constraints, and add problem specific improvements and search routine. We consider this problem to be more attractive as a benchmark comparison for general packing problems than the perfect square packing problems considered in [1–7, 12], as it requires non-trivial infeasibility proofs and contains subproblems with different amounts of slack and aspect ratio. The related square packing problem considers finding the smallest square to hold all items.

Our results show that our approach is competitive with the best known algorithms, improving the runtimes consistently by a factor of between 100 and 300. We also provide six new optimal solutions to the optimal square packing problem.

2 Constraint Programming Model

We use the established constraint model [2, 6] for the rectangle packing problem. Each item to be placed is defined by domain variables X and Y for the origin in the x, respectively y, dimension, and two integer constants W and H for the width, and respectively the height, of the rectangle. In the particular case of packing squares, W and H are identical. The constraints are expressed by a

non-overlapping constraint in two dimensions and two (redundant) CUMULATIVE constraints that work on the projection of the packing problem in x or y direction. This is illustrated by Figure 1. We use SICStus Prolog 4.0.2 [8], which provides both CUMULATIVE [1] and DISJOINT2 [3] constraints for this purpose.

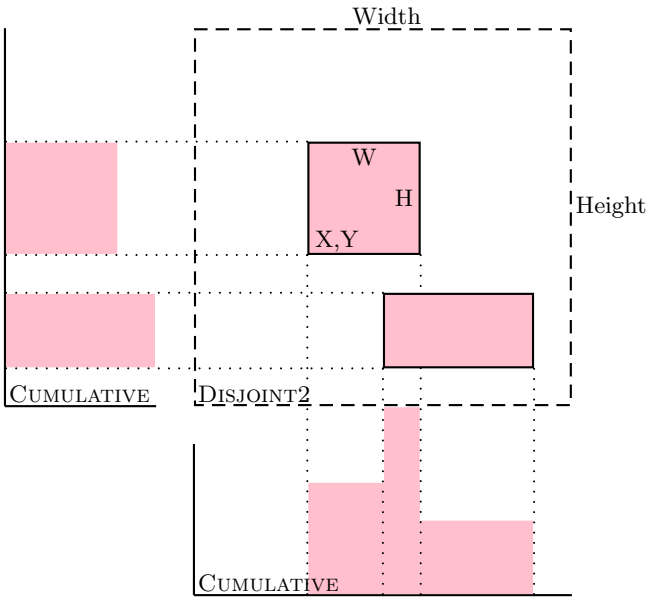

Fig. 1. The basic constraint programming model.

2.1 Decomposition

To find the enclosing rectangle with smallest area, we need a decomposition strategy that generates sub-problems with fixed enclosing rectangle sizes. As a first step we solve the square packing problem and obtain its optimal solution *UB*. We then enumerate all pairs *Width*, *Height* that satisfy

$$[\texttt{Width}, \texttt{Height}] :: n..\infty, \texttt{Width} > \texttt{Height}$$

$$\sum_{i=1}^{n} i^2 \leq \texttt{Width} * \texttt{Height} \leq \texttt{UB}^2$$

$$\texttt{Width} \geq \sum_{j=\left\lfloor \frac{\texttt{Height}+1}{2} \right\rfloor}^{n} j. \tag{1}$$

Equation 1 implements a simple bound on the required area, considering all large squares that cannot be stacked on top of each other, and which therefore must all fit horizontally.

We compute all solutions (bounding rectangles), and sort them by increasing area and increasing *Height*, i.e. for two solutions with the same surface we try the "less square-like" solution first. We then solve the rectangle packing problem for each such rectangle in turn, until we find the first feasible solution. By construction, this is an optimal solution.

Note that this decomposition approach differs from both [10] and [11]. Moffitt and Pollack do not impose a priori limits on the rectangle to be filled, while Korf builds solutions starting from an initial wide rectangle. Both methods are *anytime* algorithms, while our method only provides one initial feasible solution. Whether this distinction is important will depend on the particular intended application. Note that Korf will have to show infeasibility of the same or larger, more difficult rectangles to prove optimality, while the search space for Moffitt and Pollack looks very different.

2.2 Symmetry Removal

The model so far contains a number of symmetries, which we need to remove as we may have to explore the complete search space. We restrict the domain of the largest square of size $N \times N$ to be placed in an enclosing rectangle of size $Width \times Height$ to

$$X :: 1..1 + \left\lfloor \frac{Width - N}{2} \right\rfloor, Y :: 1..1 + \left\lfloor \frac{Height - N}{2} \right\rfloor.$$

3 Search Strategy

We have tested a number of search strategies for this problem; three methods are competitive for different sub-problem types. The *x interval* strategy of [4] first splits the domain of the x variables into intervals (in order of decreasing square size), before choosing fixed values for them. It then repeats the process for the y variables. The size of the interval is an important parameter and is fixed empirically to 0.3 times the size of each square. This method works well for non-square like problems with little or no slack and is the default used in the experiments. The *xy interval* strategy splits both x and y variables for each square into intervals, creating obligatory parts for both Cumulative and Disjoint2 constraints. It is the most stable of the methods, and works reasonable well even if there is slack in the problem. An interval size of 0.75 times the square size worked best for the problems considered. The *dual* strategy from [2, 6] chooses subsets of squares and fixes them to the left-most position possible, before fixing the y variables. This method only works if there is little or no slack in the sub-problem.

4 Model Improvements

We also consider a number of improvements to the basic model, which significantly reduce runtime and/or backtracking steps. The first idea is to ignore the 1×1 square when setting up the constraints, as it can fit in any available gap left by the other squares. Contrary to [4], where this *increased* execution times by up to a factor of 7, we find that in our case it reduces runtime significantly. This is probably due to the amount of slack already present in most subproblems, while the perfect square packing problem has no slack at all.

The second idea is an adaptation of the empty strip dominance criterion of [9] to our model. This is a problem specific symmetry breaking method that rejects some partial packings where squares are either within a certain distance from the border, or from each other. The first case can be expressed by a simple domain restriction of the x and y variables, the second needs a specialized, redundant constraint that checks the distance between facing squares. This is weaker than the pruning in [9] and reduces the number of backtracks only marginally, while increasing runtime slightly. In the experiments below only the domain restriction is applied.

5 Results

Table 1 shows the experimental results for the rectangle packing problem of sizes 17 to 25. The columns have the following meaning:

- N is the problem size;
- *Surface* is the total surface area of all squares to be packed;
- For the *Square* packing, *Size* is the length and *Area* the surface area of the minimal enclosing square; *Loss* is the spare space (as a percentage of *Surface*) required;
- For the *Rectangle* packing, K is the number of subproblems that had to be checked, *Width* and *Height* are the size of the optimal rectangle, and *Area* is its surface area; *Loss* is the spare space in the optimal rectangle as a percentage;
- *B1* and *B2* are the backtrack steps as reported by SICStus Prolog required for the square and rectangle packing, *T1* and *T2* the time (in HH:MM:SS) required, respectively.

Table 2 compares our results to the ones reported in [11]. The results for Clautiaux, Korf and BlueBlocker were obtained on a Linux Opteron 2.2GHz machine with 8Gb of RAM. Our results use SICStus 4.0.2 on a 2GHz Pentium M Linux laptop with 1Gb of memory, i.e. our hardware was slower and had less memory than the hardware used to obtain the results we compare with. The previous best time for size 25 in [10] was over 42 days, although on a significantly slower machine.

Table 1. Rectangle Placement Overview.

		Square			Rectangle					Backtrack		Time	
N	Surface	Size	Area	Loss	K	Width	Height	Area	Loss	B1	B2	T1	T2
17	1785	43	1849	3.59	5	39	46	1794	0.50	928	1837	00:00	00:00
18	2109	47	2209	4.74	13	31	69	2139	1.42	183084	29803	00:09	00:03
19	2470	50	2500	1.21	12	47	53	2491	0.85	11902	31651	00:02	00:03
20	2870	54	2916	1.60	14	34	85	2890	0.70	5716	53347	00:01	00:08
21	3311	58	3364	1.60	19	38	88	3344	1.00	16035	208947	00:04	00:27
22	3795	62	3844	1.29	15	39	98	3822	0.71	17133	886235	00:05	02:36
23	4324	66	4356	0.74	19	64	68	4352	0.65	140396	4771926	00:45	14:19
24	4900	71	5041	2.88	17	56	88	4928	0.57	378894	8278035	00:28	28:14
25	5525	75	5625	1.81	17	43	129	5547	0.40	985287	74084617	06:41	03:56:08

Table 2. Comparison with the current state-of-the-art.

N	Clautiaux	Korf	BlueBlocker	SICStus	Improvement Factor
18	31:33	1:08	1:29	0:03	22
19	72:53:18	8:15	4:11	0:03	83
20		13:32	15:03	0:08	101
21		1:35:08	1:32:01	0:27	204
22		6:46:15	4:51:23	2:36	112
23		36:54:50	29:03:49	14:19	121
24		213:33:00	146:38:48	28:14	311
25				3:56:08	

6 Incomplete Heuristics

We also considered incomplete heuristics to find good solutions for the problem and evaluated these on the square packing problem. They are based on the well-known observation that good packing solutions place the large items in the corner and on the edges of the enclosing field without any lost space. The smaller items and the slack space are used inside the packing area. We only consider one side, say the left one, of the board for our heuristic, and assume that the biggest square is placed in the bottom left corner. We then try to find combinations of $K-1$ other squares that fill the left edge completely.

We precompute all possible solutions with a small finite domain constraint program. Once all solutions are found, we order them by decreasing size of the smallest square, and use them as initial branches in our packing model, setting the x coordinate of the selected squares to 1, as well as fixing the biggest square in position $(1,1)$. Note that we do not fix the relative placement in the y direction, this is determined by the remainder of the search routine. If no solution for the given *Size* is found, we backtrack and recompute the heuristic for the next larger value.

Optimal solutions for the square packing problem up to size 25 are already known from [10]. We find six new optimal values shown in Table 3, $T_{\texttt{opt}}$ is the time required to find the optimal solution, $T_{\texttt{proof}}$ the time for the proof of optimality with the full model. A dash indicates that a lower bound is reached, thus implicitly proving optimality.

Table 3. New optimal solutions for square packing.

Problem Size	26	27	29	30	31	35
Optimal Solution	80	84	93	98	103	123
$T_{\texttt{opt}}$	12:26	00:04	11:06	2:07	00:18	1:10:07
$T_{\texttt{proof}}$	1:25:22	-	-	-	-	-

Acknowledgment

This work was supported by Science Foundation Ireland (Grant Number 05/IN/I886). The authors wish to thank Mats Carlsson, who provided the SICStus Prolog 4.0.2 used for the experiments.

References

1. A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling problems. *Journal of Mathematical and Computer Modelling*, 17(7):57 –73, 1993.
2. N. Beldiceanu, E. Bourreau, and H. Simonis. A note on perfect square placement, 1999. Prob009 in CSPLIB.
3. N. Beldiceanu and M. Carlsson. Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint. In Walsh [13], pages 377–391.
4. N. Beldiceanu, M. Carlsson, and E. Poder. New filtering for the cumulative constraint in the context of non-overlapping. In *CP-AI-OR 08*, Paris, May 2008. to appear.
5. N. Beldiceanu, M. Carlsson, E. Poder, R. Sadek, and C. Truchet. A generic geometrical constraint kernel in space and time for handling polymorphic -dimensional objects. In Christian Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 180–194. Springer, 2007.
6. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
7. N. Beldiceanu, Q. Guo, and S. Thiel. Non-overlapping constraints between convex polytopes. In Walsh [13], pages 392–407.
8. M. Carlsson et al. *SICStus Prolog User's Manual*. Swedish Institute of Computer Science, release 4 edition, 2007. ISBN 91-630-3648-7.
9. R. E. Korf. Optimal rectangle packing: Initial results. In E. Giunchiglia, N. Muscettola, and D. S. Nau, editors, *ICAPS*, pages 287–295. AAAI, 2003.
10. R. E. Korf. Optimal rectangle packing: New results. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *ICAPS*, pages 142–149. AAAI, 2004.
11. M. D. Moffitt and M. E. Pollack. Optimal rectangle packing: A meta-CSP approach. In Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey, editors, *ICAPS*, pages 93–102. AAAI, 2006.
12. P. Van Hentenryck. Scheduling and packing in the constraint language cc(FD). In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, San Francisco, USA, 1994.
13. T. Walsh, editor. *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings*, volume 2239 of *Lecture Notes in Computer Science*. Springer, 2001.