Solving Two Dimensional Strip Packing Problem with SAT solver

Fariba Khan

Introduction

Two dimensional strip packing problem (2SPP) is a discrete optimization problem. Given a set of n rectangles with height $(h_1, ..., h_n)$ and width $(w_1, ...w_n)$ and a strip of with W, the goal is to find a minimum H so that all rectangles can be orthogonally packed into W x H without any overlap. It's assumed rectangles have fixed orientation i.e. they can't be rotated. Their sides must be parallel to side of the strip; they can't overlap either. None of the rectangles' width can't be greater than width of the strip. Only integer values are allowed for w_i, h_i, W and H.

Input:

- a set of n rectangles, $R = \{r_1, ..., r_n\}$ where each r_i has a width w_i and height h_i.
- width of a strip, W

Output: minimum height, H of the strip so that all rectangles can be fit into W x H satisfying all the constraints described above along with an arrangement.

This problem is NP-hard in strong sense as one dimensional bin packing problem (1BPP) can be transformed into 2SPP. 1BPP is an well-known NP-hard problem. Multiprocess scheduling, another well-known NP hard problem, can also be seen as a particular case of 2SPP.

2SPP is an well-studied topic in computer science research. Several real world problems solving requires solving instance of 2SPP such as the cutting of rolls of paper or textile material, loading vehicle or optimal chip layout. Detailed information on two-dimensional packing problems can be found in Lodi et al. [1] and Dyckhoff et al. [2] where later focuses on cutting and packing. Besides SAt-based method, exact and incomplete method has also been used to solve 2SSP. The exact method can provide optimal solution while incomplete can't ensure optimality. However, it's difficult to get optimal solution for problem as small as 200 rectangles with either method. This motivated researcher to use SAT based method with emergence of several state-of-the-art SAT solvers.

Reduction to SAT

2SPP problem search for the minimum height H given a set of rectangles R and width of the strip, W. SAT solver determines the satisfiability of a problem. Thus we need a corresponding decision problem where the aim is to approach search problem by solving a sequence of decision problem. Let's define decision problem as **2FHSPP**, Two Dimensional Fixed Height Strip Packing Problem.

Input:

- a set of n rectangles, $R = \{r_1, ..., r_n\}$ where each r_i has a width w_i and height h_i.
- width an height of a strip, W and H respectively.

Output: Yes or No, based on whether all rectangles can be fit into the bin W x H satisfying all constraints associated with 2SPP.

2.1 2FHSPP to SAT

Now I am going to discuss how 2FHSPP can be translated into a SAT problem [reductio to SAT]. 2FHSPP can be seen as a constraint satisfaction problem with a finite set of integer variables P, domain of each variable, D and a set of constraints over a finite subset of variables. Let x_i and y_i be the lower left corners of each rectangles r_i in the arrangement. Domain of x_i and y_i are $D_{x_i} = \{k \in \mathbb{N} | 0 \leq k < W - w_i\}$ and $D_{y_i} = \{k \in \mathbb{N} | 0 \leq k < H - h_i\}$.

Here the constraints are:

1. None of the input rectangle r_i must not overlap with the strip edges.[satisfied by domain constraint]
2. Any pair of input rectangles r_i and r_j must overlap with each other i.e. $(x_i + w_i \leq x_j) \vee (x_j + w_j \leq x_i) \vee (y_i + y_i \leq y_j) \vee (y_j + y_j \leq y_i)$.

Transforming these constraints into boolean formula is done using order encoding [3]. Then SAT encoding requires four variables $lt_{i,j}, dn_{i,j}, x_{i,e}, y_{i,f}.lefti, j$ is true if r_i is somewhere at the left to the r_j. $dn_{i,j}$ is true if r_i is somewhere below to the r_j. $x_{i,e}$ means x $\leq$ e i.e. left corner of r_i is placed at less than or equal to some integer e in horizontal direction and $y_{i,f}$ means left corner of r_i is placed at less than or equal to some integer f in the vertical direction.

For each rectangle r_i, domain of e and f are $(0 \leq e < W - w_i)$ and $(0 \leq f < H - h_i)$ and following two clauses are added for each e and f due to order encoding:

$$\begin{aligned} \neg x_{i,e} \vee x_{i,e+1} \\ \neg y_{i,f} \vee y_{i,f+1} \end{aligned} \tag{1}$$

For each pair of rectangles r_i, r_j, $(i < j)$, following clauses are added to satisfy non-overlapping constraints,

$$
\begin{aligned}
& lt_{i,j} \vee lt_{j,i} \vee dn_{i,j} \vee dn_{j,i} \\
& \\
& \text{for each e, f in } (0 \leq e < W - w_i) \text{ and } (0 \leq f < H - h_i), \\
& \neg lt_{i,j} \vee x_{i,e} \vee \neg x_{j,e+w_i} \\
& \neg dn_{i,j} \vee y_{i,f} \vee \neg y_{j,f+h_i} \\
& \text{for each e, f in } (0 \leq e < W - w_j) \text{ and } (0 \leq f < H - h_j), \\
& \neg lt_{j,i} \vee x_{j,e} \vee \neg x_{i,e+w_j} \\
& \neg dn_{j,i} \vee y_{j,f} \vee \neg y_{i,f+h_j}
\end{aligned}
\tag{2}
$$

2.2 Solving 2SPP from 2FHSPP satisfiability solution

SAT encoded 2FHSPP needs to solved repeatedly to find optimal height for the 2SPP between upper bound and lower bound of the H of 2SPP problem. lower bound, lb is found by exact method and upper bound is found by either exact or incomplete method in practice [4, 5]. Few more clauses needs to added to the 2FHSPP problem to achieve this. Let's say dnh_m is boolean variable which is true if all rectangles are packed below height m, where m is an integer with domain $(lb \leq m \leq ub - 1)$. For each rectangle r_i and each height m in $(lb \leq m \leq ub - 1)$, we need one clause

$$\neg dnh_m \vee y_{i,m-h_i} \tag{3}$$

As well as, for each m in $(lb \leq m \leq ub - 1)$, we need following due to order encoding,

$$\neg dnh_m \vee dnh_{m+1} \tag{4}$$

If we gather all clauses from (1), (2), (3), (4) and add a new clause dnh_H, then solving this will give us the satisfiability of 2FHSPP with height H. Let's denote set of clauses (1), (2), (3), (4) by Ω.

$$\Omega \vee dnh_H \tag{5}$$

(5) is the 2FHSPP problem with upper bound restricted to H.

To find optimal height for 2SPP, bisection method can be used to search within the range defined by ub and lb. First, 2FHSPP is solved with H set to the middle (integer) value of the range. If it returns SAT, then upper bound is set to H. If it returns UNSAT, lower bound is set to H. This is done repeatedly with updated and more constrained range after each run until we get our optimal height.

Optimization

3.1 Reducing the Problem

Size of the encoded SAT problem grows with size of the input rectangles. If W=H, then for n rectangles, 2FHSPP will have $O(n^2)$ clauses. Thus any reduction in search space will

enhance the the SAT-based approach. Below some techniques to reduce search space of 2FHSPP are discussed [6].

Domain of the maximum rectangle, say r_m, with (w_m, h_m), can be reduced in horizontal direction from $(0 - (W - w_m))$ to $(0 - \lfloor (W - w_m)/2 \rfloor)$. If width of any rectangle r_i satisfy $w_i > \lfloor (W - w_m)/2 \rfloor$, then $lt_{i,m}$ can be set to false and all literals $lt_{i,m}$ can be removed from the SAT problem. This also permits removal of clauses containing $\neg lt_{i,m}$.

If width of two rectangles r_i and r_j satisfy $w_i + w_j > W$, then they can't be placed side by side in horizontal direction. Thus, $lt_{i,j} = false$ and $lt_{j,i} = false$. All literals $lt_{i,j}$, $lt_{j,i}$ and the clauses containing either $\neg lt_{i,j}$ or $\neg lt_{j,i}$ can be removed from SAT problem. This reduction is also valid in vertical direction.

If there are two rectangles r_i and r_j such that $w_i = w_j$ and $h_i = h_j$, then placing r_i left to r_j is equivalent to the placing r_j left to r_i. Therefore, positional relation can be fixed by assigning $lt_{j,i} = false$ thereby removing clauses including $\neg lt_{j,i}$ and adding clause $lt_{i,j} \vee \neg dn_{j,i}$ to the problem.

3.2 Reusing clauses and assumptions in SAT solving

The approach discussed in above section to solve 2SPP is incremental in the sense that Ω in (5) is common to all 2FHSPP instances corresponding to the 2SPP. Thus, reusing learned clauses and use of assumptions will increase the efficiency of the SAT solving in this case.

Reusing learned clauses When a current assignment leads to a conflict, the conflicting clause is then investigated for the source of the conflict. Each variable assignment is either done by assumption of the solver or the results of the propagation of some other variable assignment by solver's assumption. In the later case, the variable assignment by assumption that forced this propagation assignment is found by tracing backwards. This process gives the source of the conflicts i.e. the assignment of variables. A clause prohibiting this assignment is added to the SAT problem. This is called a learned clause. These clauses drive the backtracking, prohibit the same conflict to happen in future [7].

These learned clauses can be reused in the 2SPP solving. If all the clauses of a SAT problem A is included in a SAT problem B, then by lemma-resuability theorem [8], the set L of learned clauses generated in solving in A can be used to solve B. Solving B $\cup$ L instead of B alone results in reducing the search space. Approach to solve 2SPP discussed in section 2 matches the criteria of learned clause reusability i.e.learned clauses generated in solving previous 2FHSPPs can be reused in subsequent 2FHSPPS.

Reusing assumptions To solve 2SPP, unit clause dnh_H needs to be added to Ω find satisfiablity at height H as described in section 2.2, t. If it returns UNSAT, we cannot continue the bisection method before removing the dnh_H from the SAT problem. However, Eén and Sörensson [7] proposed the use of assumption to tackle this issue. The idea is to add the assumption to the problem before solving and removing it later when solving is done. This enables to continue with bisection method without recreating the SAT instance until optimal height is found.

Now this assumption can again be reused in subsequent problem. If 2FHSPP solved with assumption, say $dnh_{H'}$ returns UNSAT then, it can be added to the problem as ($\Omega \cup \neg dnh_{H'}$

) and solved again with next assumption say, $dnh_{H''}$. If second assumption returns SAT, it then can be added as $(\Omega \cup \neg dnh_{H'} \cup dnh_{H''})$ and solved with next assumption and the process continues like this to avoid redundant search space.

Performance Evaluation of the scalability of the tool

For W=H, each 2FHSPP has $O(n^2)$ clauses for n rectangles. Efficient and scalable algorithms for SAT were developed during the 2000s and have contributed to dramatic advances in our ability to automatically solve problem instances involving tens of thousands of variables and millions of constraints (i.e. clauses)[source: WIkipedia]. The SAT solver I am using is based on conflict-driven clause learning method which is one of the high performance solver for solving instances of SAT in practice. Then, with solver that can solve million clauses, theoretically it should be able solve a 2FHSPP instance with thousands rectangles. However, We need to solve several instances of 2FHSPP to solve 2SPP and time needed for solving one instance is of concern too. I need to spend more time on it to find a reasonable size of the problem given range between upper and lower bound that can be solved with described method here.

Discussion

In this section, I am going to discuss the input, output format of the tool, data sets used for evaluation, corresponding results and performance.

5.1 Tool

Tool can be downloaded from github at https://github.com/faribaK/2SPPwSATSolver.git. This tool is written in language C with Minisat as a SAT solver. It takes input as a text file format. Upon running, tool asks for a string containing path to the input file. The input file needs to be in following format:

```
number n of items
width W for the strip
for each rectangle i (i=1,...,n):
.        index i, width w_i and height h_i of item i.
lower bound (optional)
upper bound (optional)
```

Last two lines are optional. If lower and upper bound are not found in the file, tool will call appropriate function to compute them by itself.

Corresponding output will be printed in a text file with same name as input file with a suffix $'_sR_bR_out'$ in folder 'IO/outputfiles' inside Project folder. Output will contain sat problem size for the input, time required to get to the minimum height by bisection method and the minimum height found by the tool followed by the corresponding arrangement. Output file format:

PROBLEM SIZE: clauses - #clauses literals - #literals
TIME TO SOLVE: time in sec
HEIGHT: min height found by the tool (optimal/ optimal within range/)
for each rectangle i (i=1,...,n):
. (x_i, y_i) coordinate of the lower left corner of rectangle i in satisfiable arrangement

If this tool finds two consecutive unsatisfiable and satisfiable condition within the given range in given time-limit, then it reports outputs as optimal. If tool can find a satisfiable arrangement at the lower bound of the range i.e. the whole range is satisfiable, then it reports 'optimal within range' as there is no way for the tool to know if there is satisfying condition below the given lower bound. If none of the above condition satisfies, it reports best found height with nothing written besides it.

5.2 Experimental Results

All results are generated by running this tool on a Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz machine with 16GB RAM. For benchmarking, I used data sets from literature by Martello et al. [4]. All data sets are available at "DEIS - Operations Research Group Library of Instances" [9]. This benchmark set includes some problems that are very hard to solve. Lower and upper bounds for data sets are found by either exact methods [4] or incomplete methods [10] [11]. These lower and upper bounds are pretty tight and exactly same when optimal value can be found by those methods. Thus, I manually changed some of the data sets lower and upper bound to make the range larger to see whether my tool can reach the optimal value as well as to see how it performs with larger number of clauses due to larger range.

Table 1: Results on problem instances from Literature

Data	n	W	lb	ub	# clauses	#literals	Time	solution	Optimum or best known value	
									Exact [4]	Incomp. [10] [11]
HT01	16	20	20	22	6595	18446	0.04	20	20(op)	20(op)
HT02	17	20	20	23	7901	22174	3.767	20	20(op)	20(op)
HT03	16	20	19	22	6819	19088	3.11	20	20(op)	20(op)
HT04	25	40	15	16	24216	69407	54.14	15	15(op)	15(op)
HT05	25	40	15	16	25007	71770	0.66	15	15(op)	15(op)
HT06	25	40	15	16	24873	71364	0.6	15	15(op)	15(op)
HT07	28	60	30	30	48163	139846	349.28	30	**31**	30(op)
CGCUT01	16	10	23	25	6133	17120	0.03	23	23(op)	23(op)
CGCUT02	23	70	63	67	43705	126817	33.94	65	**67**	**65**
GCUT01	10	250	655	1016	86456	245066	22.2	1016	1016(op)	1016(op)
GCUT02	20	250	1208	1208	406627	1194304	1.7	1208	**1208**	**1187**
GCUT03	30	250	1803	1803	1450947	4295710	1.36	1803	1803(op)	1803(op)
NGCUT03	21	10	28	32	13349	37831	0.34	28	28(op)	28(op)
NGCUT01	10	10	19	23	1830	4970	0.11	23	23(op)	23(op)
NGCUT02	17	10	28	30	7504	21154	1.13	30	30(op)	30(op)
NGCUT03	21	10	28	32	13349	37831	0.36	28	28(op)	28(op)
NGCUT04	7	10	17	20	517	1304	0.02	17	17(op)	17(op)
NGCUT05	14	10	36	36	4729	13116	0.01	36	36(op)	36(op)
NGCUT06	15	10	29	31	5742	16088	3.15	31	31(op)	31(op)
NGCUT07	8	20	20	20	1359	3684	0.01	20	20(op)	20(op)
NGCUT08	13	20	32	38	5284	14694	0.17	33	33(op)	33(op)
NGCUT09	18	20	50	50	13790	39349	4.03	50	**50**	**50**
NGCUT10	13	30	58	80	9120	25593	0.48	80	80(op)	80(op)
NGCUT11	15	30	50	57	10072	28486	0.63	52	52(op)	52(op)
BENG01	20	25	30	33	15015	42722	0.38	30	30(op)	30(op)
BENG02	40	25	57	58	101225	295248	49.928	57	**58**	57(op)
BENG03	60	25	84	85	326513	961104	3550.09	84	**85**	84(op)
BENG04	80	25	107	109	736166	2176186	37.43	108	**108**	107(op)
BENG05	100	25	134	134	1401850	4155644	15881.12	134	134(op)	134(op)
BENG06	40	40	36	37	91460	266193	0.211	36	**37**	36(op)
BENG07	80	40	67	69	575716	1696836	12.75	68	67(op)	67(op)
BENG08	120	40	101	103	1787368	5295342	136.39	102	101(op)	101(op)
BENG09	160	40	126	134	3992471	11857979	697.15	130	126(op)	126(op)
BENG09	160	40	126	134	3992471	11857979	4588.37	127	126(op)	126(op)
BENG10	200	40	156	160	7282621	21665866	9877.98	158	156(op)	156(op)

5.3 Performance

In HT data set, optimal height can be found for all except HT07 by both exact and incomplete methods. I modified the range for almost all of them, specially for HT03, and tool successfully reached optimal height of 20 within 3.11 secs. For HT07, exact method cannot reach the

optimal height but this sat based tool can find a satisfiable arrangement for the optimal value 30 obtained by incomplete method. I didn't change range for this one for experiment due to timing constraints and also because I did this similar experiments for bigger data sets explained below. Nonetheless, if it can find satisfiable condition for 30 once, then within any range, it should be able to find it with bisection method.

Next I'll describe BENG family as I did most experiments with this one and it has some bigger data set than others. For BENG01, BENG07 and BENG08, this tool reached optimal value within modified large range. For BENG02, BENG03 and BENG06, exact method can't find optimal value but this tool reached optimal height (found by incomplete method) for all of them. For BENG03, it took 3550 secs to find satisfiability for optimal value although number of clauses are less than GCUT02 and GCUT03 where it took less than 2 secs to satisfy for optimal. This is due to the intrinsic hardness of the problem which leads to more conflicts even with size-able sat problem. More rectangle in BENG03 is also a notable factor too but hardness of the problem doesn't only depend on size of the input. More conflicts makes sat solver to propagate backwards more often to try more assignments. This also generates larger set of learned clauses. Following experiment strengthen this observation. BENG05 with 100 rectangles took huge amount of time to satisfy for optimal value although its sat problem size and used range are less than BENG10 (200 rectangles)'s experiment which took less time to satisfy for a value (158) closer to optimal (156). BENG10 got timeout at 3600*6 secs when trying to solve for 157 and reported best found value 158. For BENG09 with optimal value 126, this tool reached 130 in 697 secs but took 4588 secs to solve for 127. Thus for harder problem, trying to get close to optimal value takes more resources.

For NGCUT family, this tool reached optimal value for all of them except for NGCUT09 where optimal value can't be found by other two methods either. Most of them are processed within a fraction of 1 secs with only one exceeding 3secs. NGCUT09 is apparently a hard problem and this tool reached best known value 50 within 4 secs. I didn't venture into finding satisfialbility or unsatisfiability for 49.

For CGCUT02, neither exact nor incomplete method can reach optimal value. Lower bound found by exact method is however 63. Given range 63-67, this tool reached 65, which is the best of other two methods, within 34 secs. It got timeout at 3600 secs while trying to find satisfiable solution for 64, I didn't try this one further with longer time limit. For GCUT02, this tool reached best solution of exact method in 1.7 secs. I didn't get time to try it for bigger range.

Conclusion

Form discussion in above section 5.3, it can be observed for the scalability of the tool that performance of the tool i.e. execution time doesn't only depend on input size of the problem. Relative sizes and combinations of rectangles affect the size of the sat problem. GCUT02 and GCUT03 with only 20 and 30 rectangles requires solving sat problem of size comparable to BENG08 and BENG09 with 120 and 160 rectangles respectively. HT07 with 28 rectangles requires only 0.6 secs to satisfy for optimal. Again, bigger sat problems can be solved faster if there are less conflicts like mentioned above. Thus, it's very hard to come up with some upper size limit for the input solvable in this tool without considering the nature of the problem.

However, during experiment, I saw problem can't be solved in reasonable amount of time (6-7 hours) when number of conflict exceeds 10^8 which happened in BENG09's experiment when trying to solve for optimal value of 126. However, when number of rectangles is small overall time to solve is less than the bigger input, thus can be said, this tool is more promising in finding optimal value for them even for hard problems. Also, sat based method seems to perform better than exact method in terms of finding optimal and is comparable to incomplete method.

References

[1] A. Lodi, S. Martello, and D. Vigo, "Recent advances on two-dimensional bin packing problems," *Discrete Appl. Math.*, vol. 123, no. 1-3, pp. 379–396, Nov. 2002.

[2] G. S. J. T. Dyckhoff, H., "Cutting and packing (c'—&'p)," *Annotated Bibliographies in Combinatorial Optimization, M. Dell'Amico, F. Maffioli, S. Martello (eds.)*, pp. 393–412, 1991.

[3] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara, "Compiling finite linear csp into sat," *Constraints*, vol. 14, no. 2, pp. 254–272, Jun 2009.

[4] S. Martello, M. Monaci, and D. Vigo, "An exact approach to the strip-packing problem," *INFORMS J. on Computing*, vol. 15, no. 3, pp. 310–319, Jul. 2003.

[5] E. Gimadi, V. Zalyubovskiy, and P. Sharygin, "The problem of strip packing: An asymptotically exact approach," *Russian Mathematics*, vol. 41, 01 1997.

[6] T. Soh, K. Inoue, N. Tamura, M. Banbara, and H. Nabeshima, "A sat-based method for solving the two-dimensional strip packing problem," *Fundam. Inf.*, vol. 102, no. 3-4, pp. 467–487, Aug. 2010.

[7] N. Eén and N. Sörensson, "An extensible sat-solver," E. Giunchiglia and A. Tacchella, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 502–518.

[8] H. Nabeshima, T. Soh, K. Inoue, and K. Iwanuma, "Lemma reusing for sat based planning and scheduling," in *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'06. AAAI Press, 2006, pp. 103–112.

[9] $http://www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm$.

[10] R. Alvarez-Valdes, F. Parreño, and J. M. Tamarit, "Reactive grasp for the strip-packing problem," *Comput. Oper. Res.*, vol. 35, no. 4, pp. 1065–1083, Apr. 2008.

[11] L. Wei, D. Zhang, and Q. Chen, "A least wasted first heuristic algorithm for the rectangular packing problem," *Comput. Oper. Res.*, vol. 36, no. 5, pp. 1608–1614, May 2009. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2008.03.004