Shikaku as a Constraint Problem

Helmut Simonis*

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

Abstract. In this paper we describe models for the Shikaku puzzle using CP, MIP and SAT techniques. A finite domain constraint model is used to generate the SAT or MIP set partitioning problems to be solved. Results on some examples indicate that this is a very succesful decomposition method for this puzzle, all examples are solved in less than two seconds.

1 Introduction

Shikaku is a logic puzzle from the Japanese puzzle company Nikoli [15]. Figure 1 shows a simple example from the Nikoli website [14]. The rules of puzzle are:

9				12			5		
									6
8		6		8					
					6		8		12
4									
		3			9				4

Fig. 1. Example Problem

1. The puzzle is played in a given recti-linear grid. Some grid cells contain numbers.
2. The task is to partition the grid area into rectangular *rooms* satisfying the conditions:

* This work was supported by Science Foundation Ireland (Grant Number 05/IN/I886). Support from Cisco Systems and the Silicon Valley Community Foundation is gratefully acknowledged.

(a) Each room contains exactly one number.
(b) The area of the room is equal to the number in it.

The partitioning means that rooms can not overlap, and every cell in the grid must belong to a room.

As is usually the case with this type of logic puzzles, the problem has a unique solution.

Figure 2 shows the solution of the example.

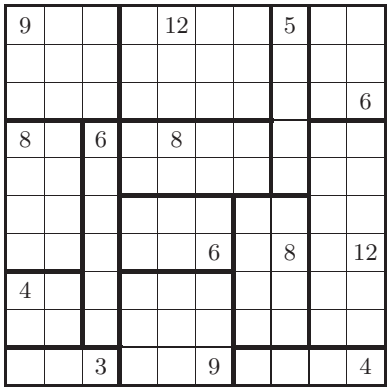

Fig. 2. Example Solution

2 Related Work

Placing a set of objects in a larger space without overlap is the objective of many puzzles. Shikaku is special as we only use rectangles in the grid, and the rooms can only be placed close to their hint. This restricts the complexity of the problem significantly.

An example of solving the Shikaku puzzle with the *geost* non-overlapping constraint is shown in [2].

The constraint model for a similar problem called perfect square placement was discussed in [3], using *diffn* [4] and *cumulative* [1] constraints in CHIP. This extends earlier work in [1] and [7]. The report [3] is available as problem 009 in the csplib.

Another non-overlapping rectangle problem is described in [9]. The objective is to place a set of items in the smallest enclosing rectangle. This is a much harder problem, as the aspect ratio of the enclosing rectangle is not known.

3 Model

An obvious model to solve this problem would use the *diffn* constraint to express non-overlapping of rectangles in the given grid area. Unfortunately, ECLiPSe does not provide a diffn constraint. A model which breaks the non-overlap into primitive constraints seems quite weak, so that we choose an alternative approach.

The main idea is to decompose the problem solving into two sequential steps. In a first step, we generate possible placements for each room with a small finite domain program. This creates all possible rectangles within the grid which cover one hint, do not include the other hints, and have the right area.

The second step then selects a subset of these pattern so that every grid point is covered by exactly one pattern. As we also do this for the grid cell containing hints, we guarantee that each hint is covered by a room of the correct size. This second step is a set partitioning problem. The decomposition of a larger problem into a pattern generation and a subsequent set partitioning (or covering) step is well known for many large scale transportation problems.

3.1 Placement Alternatives

The following program is used to create the possible room placements for a hint in location I, J with area N and id K. *Other* contains the list of all other hints, *Width* and *Height* are the grid size. A possible placement for the room is located at position X, Y, it has width W and height H. The following constraints are imposed:

- The product $W * H$ must be equal to the area N.
- The placement must fit into the grid.
- The location I, J must be inside the placement.
- All other hints must be outside the placement.

```
rectangle(hint(I,J,N,K),Others,Width,Height,rectangle(X,Y,W,H,N,K)):-
        ic:(X :: 1..Width),
        ic:(Y :: 1..Height),
        ic:(W :: 1..N),
        ic:(H :: 1..N),
        W*H #= N,
        X+W-1 #=< Width,
        Y+H-1 #=< Height,
        inside(X,Y,W,H,I,J),
        outsides(X,Y,W,H,Others),
        search([X,Y,W,H],0,input_order,indomain,complete,[]).
```

The constraint *inside* checks that I, J is inside the rectangle spanned by X, Y and $X + W, Y + H$.

```
inside(X,Y,W,H,I,J):-
        I #>= X,
        J #>= Y,
        I #< X+W,
        J #< Y+H.
```

The constraint *outside* checks that I, J is outside the rectangle, *outsides* iterates over all hints to impose the constraint on all hints.

```
outsides(X,Y,W,H,L):-
        (foreach(hint(I,J,_,_),L),
         param(X,Y,W,H) do
            outside(X,Y,W,H,I,J)
        ).

outside(X,Y,W,H,I,J):-
        I #< X or J #< Y or I #>=X+W or J #>= Y+H.
```

3.2 Partitioning

Once all possible placements are generated, we can check the partitioning condition. Every grid cell must be covered by exactly one selected placement. Let G be the set of grid points $< i, j >$ in the problem and U the set of rooms to be created. Then let R be the set of possible placements as tuples $< x, y, w, h, u >$. We use $0/1$ integer variables X_r which state that possible placement r is selected. Then we can impose a constraint for each grid cell

$$\forall_{<i,j>\in G}: \sum_{r\in R, r\texttt{overlap}<i,j>} X_r = 1$$

We can solve this set of equations with either MIP, a Pseudo-Boolean solver or a $0/1$ finite domain model. In our evaluation we use both the finite domain solver as well as the default eplex [8] solver of ECLiPSe [17], which uses OSI clpcbc as the underlying LP/MIP platform. We also tested Minisat+ [6], an efficient Pseudo-Boolean solver based on the SAT solver Minisat [5] with similar results.

3.3 Non-Overlap Binary Model

An alternative way of expressing the constraints uses a direct SAT expression of the non-overlap constraint. We again use variables X_r which indicate if placement r is used or not. For two placements r_1 and r_2 that overlap, we state a clause

$$\forall_{r_1,r_2\in R, r_1\texttt{overlaps}r_2}: \quad \neg X_{r_1} \vee \neg X_{r_2}$$

In addition we require that at least one of the placements for a given room u is used, this introduces clauses

$$\forall_{u\in U}: \bigvee_{r_k\texttt{covers}u} X_{r_k}$$

for all rooms U.

Note that this model is implied by the partitioning equalities, but is strictly weaker. We would have to add

$$\forall_{<i,j>\in G} : \bigvee_{r_k \texttt{overlap} <i,j>} X_{r_k}$$

for all grid points which do not contain hints to make them equivalent. For the given examples, this does not seem to make any difference in performance.

4 Results

Tables 1 (FD model) and 2 (MIP model) show the results for a number of example puzzles from the Nikoli website [14] and some of their puzzle collections [11, 10, 12, 13, 16].

Each line reports the result for one puzzle, *Set* is the collection from which it is taken, *Nr* the number in the collection, *Grade* shows the grade given by Nikoli, X and Y are the grid size. *Rooms* gives the number of rooms on the grid, this is equal to the number of hints on the puzzle. *Fixed* shows how many rooms are fixed, i.e. there is only one possible placement for it. The total number of placement alternatives is given by *Alt*. The column *Cover* shows the number of grid locations only covered by one pattern. *Time* is the total time to generate the model and solve the formulation. *Nodes* is the number of nodes explored by the solver (This is possibly incorrect for eplex). In table 1 we have in addition *Setup*, the number of variables left after initial propagation, and *Shave*, the number of variables left after a single shaving run over all variables. In table 2 we also report *Step1*, the time needed to run the first part of the problem solver, up to the point where the covers for each grid cell are generated.

The results show that all problems are solved in less than 2 seconds, and require very little (if any) search. In the FD model, half of the problems are solved by propagation in the model alone, the other half requires shaving to avoid search. Comparing *Setup* and total run time *Time* we see that execution time is dominated by the first part of the model.

5 Summary

In this paper we have described some models for the Shikaku puzzle, a rectangle partitioning problem in a recti-linear grid. By decomposing the problem into two steps, good results are achieved on all given examples. In the first step, all possible placement alternatives are generated by a finite domain program, which takes restrictions on the placement into account. In a second step a set partitioning model is used to find the minimal cover of the grid with a MIP, SAT, Pseudo-Boolean or CP model. This decompostion is well known for problems like aircraft rotation planning, crew diagramming and other transportation problems, and works very well for this problem.

Table 1. FD Result Summary

Set	Nr	Grade	X	Y	Rooms	Fixed	Alt	Cover	Setup	Shave	Time	Nodes
nikoli-web	0	easy	10	10	20	3	80	6	0	0	0.06	0
nikoli-web	1	easy	10	10	14	2	81	5	0	0	0.02	0
nikoli-web	2	easy	10	10	16	1	83	2	0	0	0.02	0
nikoli-web	3	easy	10	10	28	3	96	12	0	0	0.03	0
nikoli-web	4	easy	10	18	44	15	78	78	0	0	0.07	0
nikoli-web	5	medium	10	18	38	3	115	21	0	0	0.06	0
nikoli-web	6	medium	10	18	36	1	210	3	153	0	0.07	0
nikoli-web	7	medium	14	24	68	4	293	16	0	0	0.17	0
nikoli-web	8	hard	14	24	70	8	380	17	0	0	0.19	0
nikoli-web	9	hard	20	36	128	4	872	8	196	0	0.58	0
nikoli-web	10	hard	20	36	120	1	986	1	774	0	0.68	0
giant	0	easy	7	7	12	2	46	2	0	0	0.01	0
giant	1	-	35	21	149	0	1002	2	995	0	0.76	0
giant	2	-	35	21	189	0	888	5	855	0	0.89	0
giant	3	-	35	21	209	5	1023	6	830	0	1.07	0
giant	4	-	35	21	190	2	849	8	495	0	0.90	0
giant	5	-	35	21	137	1	990	3	929	0	0.67	0
giant	6	-	35	21	160	2	900	5	323	0	0.76	0
giant	7	-	35	21	135	4	1111	4	984	0	0.74	0
giant	8	-	35	21	140	1	979	3	943	0	0.85	0
giant	1701	hard	45	31	255	10	1731	25	617	0	1.93	0
giant	1702	hard	45	31	206	3	1727	11	1497	0	1.95	0
giant	1801	hard	45	31	189	12	1468	35	0	0	1.45	0
giant	1802	hard	45	31	276	11	1517	34	769	0	2.10	0
giant	1901	hard	45	31	188	8	1443	36	148	0	1.48	0
giant	1902	hard	45	31	213	4	1813	11	1449	0	1.82	0
giant	2001	hard	45	31	221	16	1217	92	0	0	1.51	0
giant	2002	hard	45	31	278	6	1617	20	1259	0	2.21	0

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. Generalised spatial and temporal placement constraint: Current status and evolution. In *Eurocontrol Innovative ATM Research Workshop - Constraint Programming for ATM*, December 2008.
3. N. Beldiceanu, E. Bourreau, and H. Simonis. A note on perfect square placement. Technical report, COSYTEC, 1999. prob009 in CSPLIB.
4. N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
5. Niklas En and Niklas Srensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

Table 2. MIP Result Summary

Set	Nr	Grade	X	Y	Rooms	Fixed	Alt	Cover	Step1	Time	Nodes
nikoli-web	0	easy	10	10	20	3	80	6	0.05	0.05	0
nikoli-web	1	easy	10	10	14	2	81	5	0.01	0.02	0
nikoli-web	2	easy	10	10	16	1	83	2	0.02	0.03	0
nikoli-web	3	easy	10	10	28	3	96	12	0.03	0.03	0
nikoli-web	4	easy	10	18	44	15	78	78	0.06	0.07	0
nikoli-web	5	medium	10	18	38	3	115	21	0.05	0.06	0
nikoli-web	6	medium	10	18	36	1	210	3	0.06	0.08	0
nikoli-web	7	medium	14	24	68	4	293	16	0.16	0.19	0
nikoli-web	8	hard	14	24	70	8	380	17	0.17	0.21	0
nikoli-web	9	hard	20	36	128	4	872	8	0.55	0.64	0
nikoli-web	10	hard	20	36	120	1	986	1	0.55	0.82	0
giant	0	easy	7	7	12	2	46	2	0.01	0.01	0
giant	1	-	35	21	149	0	1002	2	0.71	0.90	0
giant	2	-	35	21	189	0	888	5	0.88	0.97	0
giant	3	-	35	21	209	5	1023	6	1.01	1.16	0
giant	4	-	35	21	190	2	849	8	0.88	0.96	0
giant	5	-	35	21	137	1	990	3	0.61	0.76	0
giant	6	-	35	21	160	2	900	5	0.75	0.85	0
giant	7	-	35	21	135	4	1111	4	0.65	0.92	0
giant	8	-	35	21	140	1	979	3	0.67	0.87	0
giant	1701	hard	45	31	255	10	1731	25	1.86	2.11	0
giant	1702	hard	45	31	206	3	1727	11	1.69	2.08	0
giant	1801	hard	45	31	189	12	1468	35	1.40	1.57	0
giant	1802	hard	45	31	276	11	1517	34	2.05	2.24	0
giant	1901	hard	45	31	188	8	1443	36	1.41	1.60	0
giant	1902	hard	45	31	213	4	1813	11	1.62	2.12	0
giant	2001	hard	45	31	221	16	1217	92	1.46	1.58	0
giant	2002	hard	45	31	278	6	1617	20	2.08	2.36	0

6. Niklas En and Niklas Srensson. Translating pseudo-boolean constraints into sat. *Journal on Satisability, Boolean Modeling and Computation*, 2:1–26, 2006.
7. P. Van Hentenryck. Scheduling and packing in the constraint language cc(FD). In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann Publishers, 1994.
8. Kish Shen and Joachim Schimpf. Eplex: Harnessing mathematical programming solvers for constraint logic programming. In Peter van Beek, editor, *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 622–636. Springer, 2005.
9. Helmut Simonis and Barry O'Sullivan. Search strategies for rectangle packing. In Peter J. Stuckey, editor, *CP*, volume 5202 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2008.
10. various. *Puzzle the Giants 17*. Nikoli, 2004. In Japanese.
11. various. *Puzzle the Giants Vol 1-6*. Nikoli, 2004. In Japanese.
12. various. *Puzzle the Giants 18*. Nikoli, 2005. In Japanese.
13. various. *Puzzle the Giants 19*. Nikoli, 2006. In Japanese.

14. various. Nikoli shikaku example problems, 2007. http://www.nikoli.com/en/puzzles/shikaku/.
15. various. Nikoli web site, 2007. http://www.nikoli.co.jp/en.
16. various. *Puzzle the Giants 20*. Nikoli, 2007. In Japanese.
17. M. Wallace, S. Novello, and J. Schimpf. ECLiPSe : A platform for constraint logic programming. *ICL Systems Journal*, 12(1), May 1997.