

# Automatic Making of Sokoban Problems

Yoshio MURASE<sup>1</sup>, Hitoshi MATSUBARA<sup>2</sup> and Yuzuru HIRAGA<sup>1</sup>

<sup>1</sup> University of Library and Information Science, 1-2 Kasuga, Tsukuba, Ibaraki, 305  
JAPAN

<sup>2</sup> Electrotechnical Laboratory,  
1-1-4 Umezono, Tsukuba, Ibaraki, 305 JAPAN

**Abstract.** This paper describes our program that makes *Sokoban* problems automatically. *Sokoban* is one of one-person puzzles invented in Japan. The program consists of three stages: generation, checking and evaluation. First, candidates for problems are generated randomly by a prototype and three templates. Second, unsolvable candidates are removed by the *Sokoban* solver. Finally trivial or uninteresting candidates are removed by the evaluator. The problems that the program made are judged good by human experts. Creation of art by computer is an important target of Artificial Intelligence. Our work can be characterized one of the attempts to create some arts by computers.




## 1 Introduction

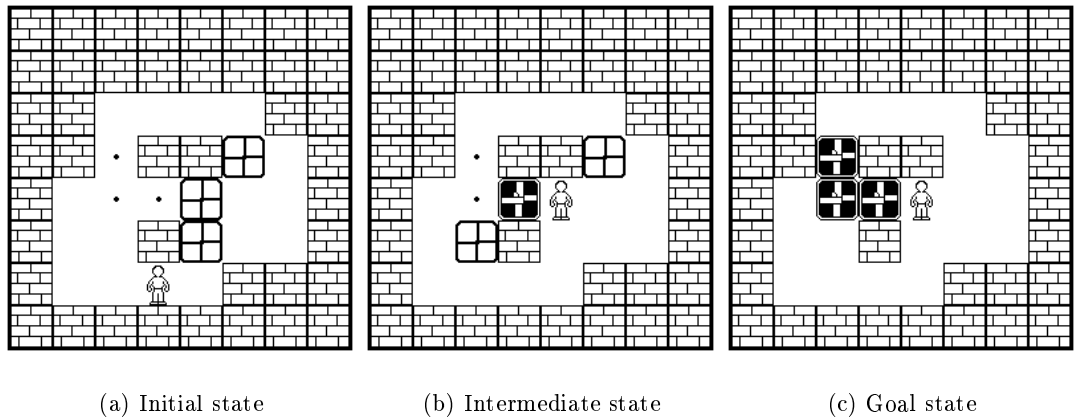
This paper describes our program that makes *Sokoban* problems automatically. Our program consists of three stages: generation, checking and evaluation. First, candidates for problems are generated with a prototype and three templates at random. Second, unsolvable candidates are removed by our *Sokoban* solver. Finally trivial or uninteresting candidates are removed by our evaluator. The problems that our program made are judged good by human experts. Creation of art by computer is an important target of artificial intelligence. Our work can be characterized one of the attempts to create some arts by computers.

## 2 Sokoban

*Sokoban* is one of one-person puzzles invented in Japan at 1982. *Sokoban* is a Japanese word meaning a warehouse-keeper. The initial state of Fig.1 is an example of a problem. A “warehouse” consists of “walls” and “passways”. The passway is a connected area of unit squares. A unit square can be occupied by either “warehouse-keeper”, an “object”, or can be empty. A number of squares (equally the number of objects) are designated as “goal areas”. The objective of the puzzle is to operate the warehouse-keeper to push all objects to goal areas (see the goal state of Fig.1). In the later part of this paper we use the following items:

–  : wall

- $\cdot$  : goal area
-  : object
-  : warehouse-keeper
-  : place where goal and object are overlapped



**Fig. 1.** An example of a Sokoban problem

There are two kinds of operations: MOVE and PUSH.

- MOVE: move the keeper one step toward four directions on the passway: up, down, right and left.
- PUSH: move an object one step forward by making the keeper push it from behind. The keeper can push only one object at a time.

While the rules are simple, the problems can get to be quite hard and interesting. *Sokoban* problems [Thinking Rabbit 1982, 1989] is one of the bestseller game softwares in Japan. *Xsokoban*[Myers 1995], a software of *Sokoban* on X-Windows, is played worldwide (in more than 20 countries so far).

Making good (or artistic) and interesting problems is a matter of expertise. Several human experts of *Sokoban* make good problems to be solved. Our objective is to make a program that makes good problems comparable to those of human experts.

Our program consists of three stages: generation, checking and evaluation. The latter sections describes these stages by showing an example. In this paper,

we fix the size of warehouse  $8 \times 8$  units and the number of objects to three. The actual program is capable of making larger and more complex *Sokoban* problems.

### 3 Generation

In this stage *Sokoban* problems are generated.

A generator is expected to keep the following conditions:

1. generate original problems.
2. generate problems that have solutions.
3. generate interesting problems.

It is impossible to keep the conditions completely, but we can improve possibility that good problems are generated by implementing several devices.

The generation stage starts from a prototype ( Fig.2 (a)). In the following example we have chosen three templates ( Fig.2 (b)).

Our program places the templates at random on the prototype. In the example, the first template is at the upper-left ( Fig.2 (c)), the second is placed at the upper-right (Fig.2 (d)), and the third is placed at the lower-right (Fig.2 (e)).

The size of all templates shown in Fig.2 (b) is more than  $2 \times 2$ . If we use a template whose size is  $1 \times 3$ , it is likely that a lot of uninteresting problems will be generated. The position of the templates are chosen to overlap existing passways, so the generated passway is guaranteed to be connected.

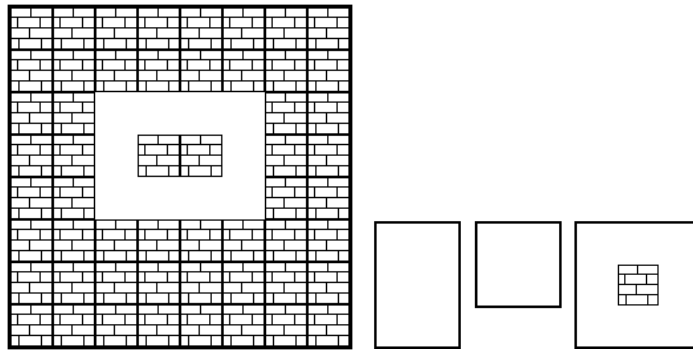
Next, goal areas are chosen on the passway. Areas to where the keeper cannot push his object are marked GOAL\_AVOID (Fig.3 (a)). Goal areas are chosen randomly avoiding GOAL\_AVOID (Fig.3 (b)).

Then objects are placed on the passway. A “dead end area” is an area in the passway from which objects cannot be moved out of. Dead end must be removed from the candidates of the initial positions of objects provided that there are no goal areas within them.

Goal areas which are at corner are marked COR\_GOAL (Fig.3 (c)). On the assumption that there is only one object, areas where the keeper can push the object from each goal area are calculated (Fig. 4). The calculated areas are marked GOAL\_RANGE.

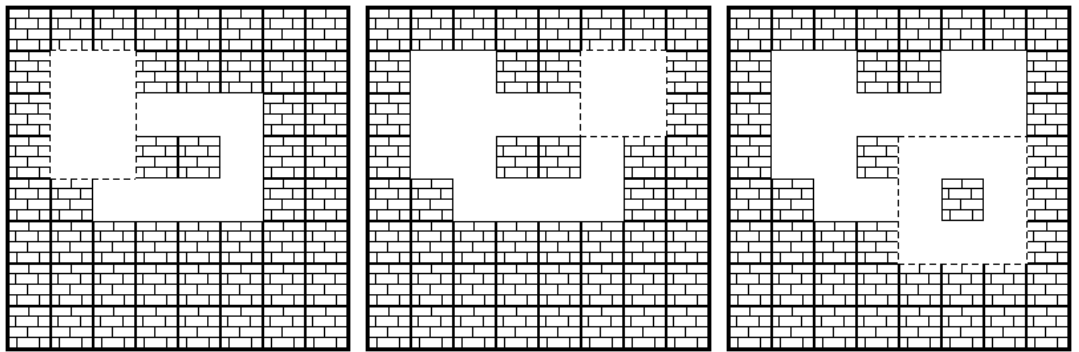
The first object is placed on an area that is in GOAL\_RANGE areas and is not in COR\_GOAL areas. New dead end areas (ADD\_AVOID areas) are calculated. The second object is placed on an area that is in GOAL\_RANGE areas and is not in COR\_GOAL and is not in ADD\_AVOID areas. New dead end areas are calculated. These steps are iterated until all the objects are placed.

In this example setting of objects are done as Fig.5. Finally the warehouse-keeper is placed on the passway at random. Fig.5 (f) is a generated problem in this example.



(a) Prototype

(b) Templates



(c) First template placed

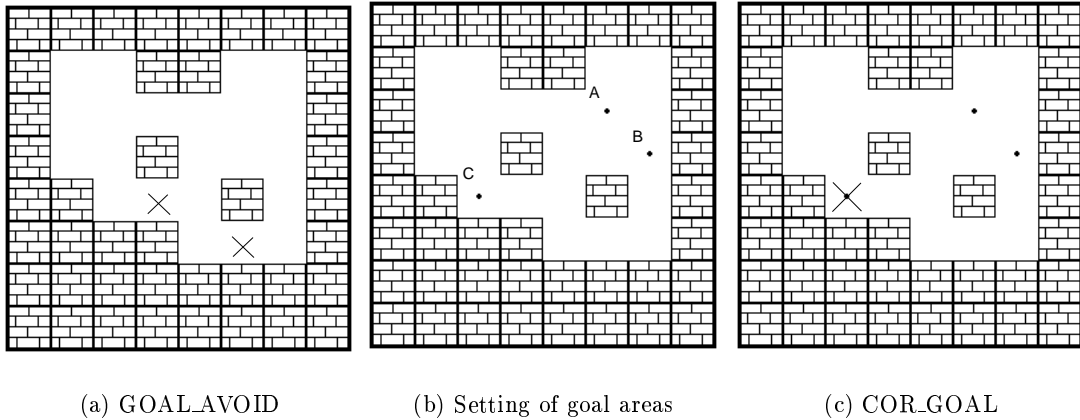
(d) Second template placed

(e) Third template placed

**Fig. 2.** Outline generation

## 4 Checking

Some of the problems generated by our program are solvable (i.e. the warehouse-keeper can push all the objects to the goal areas of the problem) while others are not. Unsolvable problems are *illegal* and must be rejected. Ueno et al. built an automatic *Sokoban* solver[Ueno, Nakayama and Hikita 1994]. which searches a solution sequence by best-first search. So it can sometimes solve problems that have long solution sequences, but it sometimes cannot solve problems that have short solution sequences. The best-first search is not adequate for checking solvability of generated problems. So we rewrote Ueno's solver into breadth-first search version. Our solver cannot solve problems that have long solution



**Fig. 3.** Setting of goal areas

sequences but it can solve all problems that have short solution sequences.

Our program check all the generated problems by the new solver. If the new solver cannot solve a generated problem, it is removed from the candidate list. In average about half of the generated problems are removed at this stage (experimental data are shown later).

## 5 Evaluation

The remaining problems are all *legal*. But most of the problems are trivial and uninteresting. The evaluation program evaluates artistic values of the remaining problems. The evaluation criteria are:

1. length of solution sequence: if a problem has a very short solution sequence, it is trivial and is rejected. when the length of a solution sequence of a problem is less than seven, the problem is rejected.
2. the number of changes in directions of pushing in solution sequence: in interesting problems the keeper has to change the directions of pushing objects very often. if the number of the changes are less than four, it is trivial and must be removed.
3. the number of detours in solution sequence: if a solution sequence of a problem has no detours, it is uninteresting and is rejected.

The above criteria removes trivial or uninteresting problems from the candidate lists. In average four fifths of the remaining problems are removed.

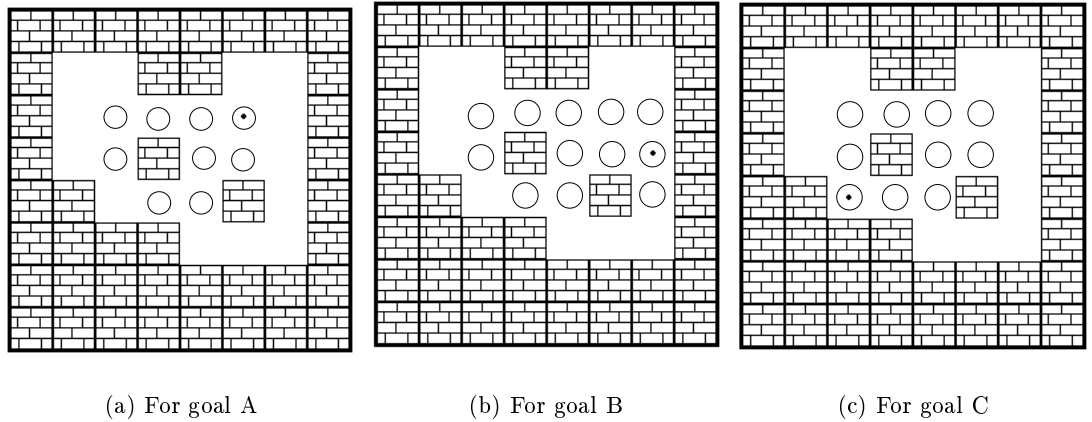


Fig. 4. Calculation of GOAL\_RANGE

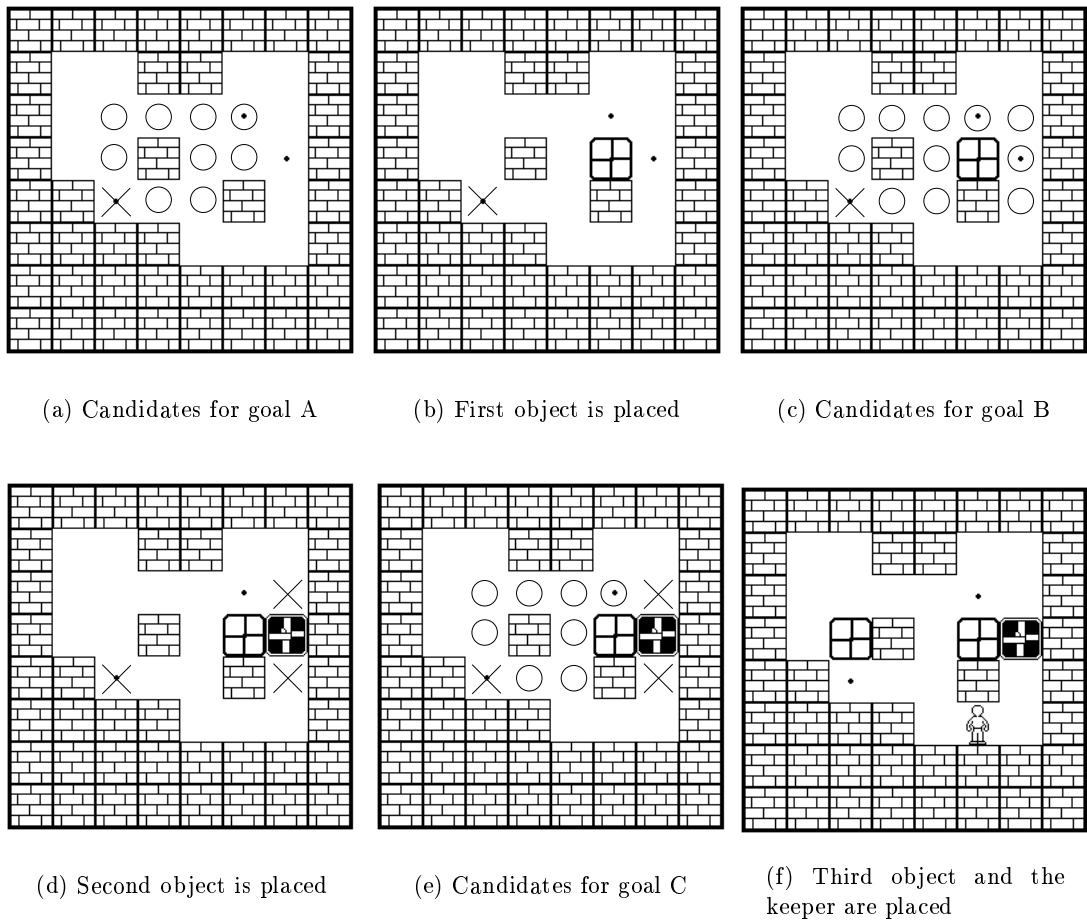
## 6 Problems that our program made

In this section we show four problems that our program made (see Fig.6). From the point of human experts' view, these problems have some artistic values.

We made experiments five hundred times with the prototype (Fig.2 (a)) and the templates set (Fig.2 (b)). The results are shown in Table 1. The generation stage failed seven times. About half of generated problems were unsolvable. Our programs outputted 44 problems as good problems, but "real" good problems which are evaluated good by human experts were fourteen out of them.

Table 1. Experimental data

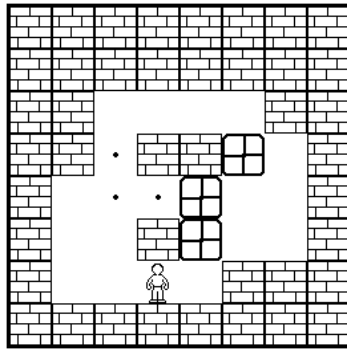
# of generation failures		7
# of problems that have no solutions		245
solvable problems	removed	204
	outputted	44
# of trials		500
real good problems		14



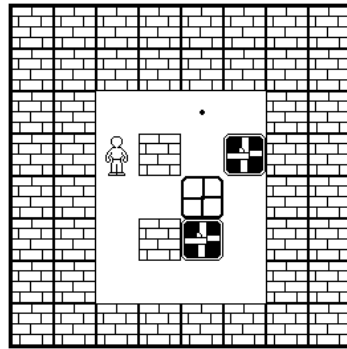
**Fig. 5.** Setting of the objects and the keeper

## 7 Concluding Remarks

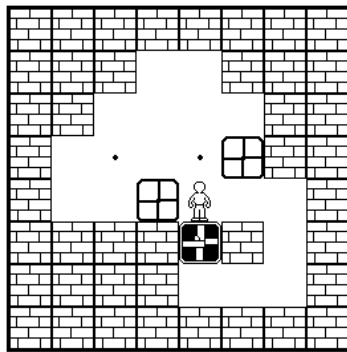
We have built a program that makes *Sokoban* problems automatically [Murase 1996]. Some problems that our program made are judged good by human experts. Our program, however, has several drawbacks. For example, our program cannot make problems that have very long solution sequences. We have to revise our program to get more complex problems.



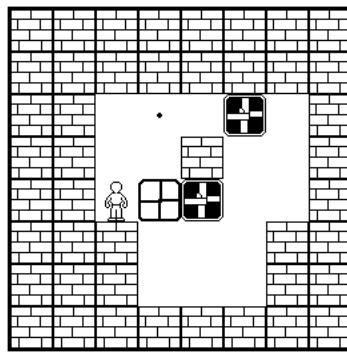
(a) Good Problem 1



(b) Good Problem 2



(c) Good Problem 3



(d) Good Problem 4

**Fig. 6.** Good problems

## References

- [Thinking Rabbit 1982, 1989] ThinkingRabbit: Sokoban problems 1,2, perfect (1982,1989)
- [Myers 1995] A. Myers: XSokoban, <http://clef.lcs.mit.edu/~andru/xsokoban.html> (1995)
- [Ueno, Nakayama and Hikita 1994] A.Ueno, K.Nakayama and T. Hikita: A program that solves *Sokoban* problems (in Japanese), bit, vol.26, no.12, pp.40-51 & vol.27, no.1, pp.92-100, Kyoritsu-shuppan, Tokyo (1994)



[Murase 1996]

Y. Murase: An attempt of automatic creation of Sokoban problems (in Japanese), Bachelor Thesis of University of Library and Information Science, Tsukuba, Japan (1996)