FI MU

Faculty of Informatics
Masaryk University Brno

Human Problem Solving: Sokoban Case Study

by

Petr Jarušek
Radek Pelánek

FI MU Report Series FIMU-RS-2010-01

Human Problem Solving: Sokoban Case Study

Petr Jarušek Radek Pelánek
Faculty of Informatics
Masaryk University Brno, Czech Republic

April 13, 2010

Abstract

We describe a case study in human problem solving for a particular problem – a Sokoban puzzle. For the study we collected data using the Internet. In this way we were able to collect significantly larger data (2000 problems solved, 780 hours of problem solving activity) than in typical studies of human problem solving. Our analysis of collected data focuses on the issue of problem difficulty. We show that there are very large differences in difficulty of individual Sokoban problems and that these differences are not explained by previous research. To address this gap in understanding we describe an abstract computational model of human problem solving, a metric of a problem decomposition, and formalization of a state space bottleneck. We discuss how these concepts help us understand human problem solving activity and differences in problem difficulty.

1 Introduction

Human problem solving is influenced by many parameters, e.g., formulation of rules, number of items that need to be held in working memory, or familiarity with problem domain. But even when all of these intuitive parameters are fixed, there can still be large differences in problem difficulty. We study human behaviour during Sokoban problem solving and we show that even for a set of very similar problems the differences in difficulty are significant and not explained by any simple problem parameter. Why do these differences occur?

Understanding this phenomenon is important not just for providing insight into human cognition, but it also has several applications. Humans and computers solve problems in different ways and have complementary strengths [22]; to built tools for human-computer collaboration we need to understand what makes problems difficult for humans. Another application is for teaching and training (e.g., with intelligent tutoring systems [3]). Humans enjoy problem solving, but only when faced with problems of adequate difficulty, i.e., neither too boring nor too difficult. To recommend appropriate instance of problem we need to be able to evaluate its difficulty.

1.1 Human Problem Solving

Problem solving encompasses many different activities. The main classification is into well-structured problems and ill-structured problems [23]. Well-structured problems have clear boundaries and sharply defined situations, rules, and goals (e.g., proving mathematical statement, optimizing logistic operation). Ill-structured problems are defined more vaguely (e.g., writing a book).

A typical example of well-structured problem are logic puzzles. Puzzles contain all important information in the statement of the problem (and hence do not depend on knowledge), are amenable to automated analysis, and are also attractive for humans. The use of puzzles has a long tradition both in computer science (particularly artificial intelligence) [20] and cognitive psychology [21, 23].

Human problem solving has been studied for a long time, starting by a seminal work by Simon and Newell [21]; for a recent overview see [17]. Relevant to this work is particularly research concerned with puzzles which can be directly expressed as state space traversal, e.g., Tower of Hanoi puzzle [14], river crossing problems [11], Water jug puzzle [4], Fifteen puzzle [18], and Chinese ring puzzle [15].

This work is concerned with analysis of human problem solving activity on the Sokoban puzzle with the special focus on the issue of problem difficulty. As opposed to previous research, which has been based only on small samples of human problem solving activity in laboratory setting, we employ large scale data collection through Internet. The data are collected via web portal which contains a simulator of the puzzle. People log into this portal and try to solve provided puzzles. All their actions (including their timing) are saved and stored into a database on the server.

This approach has certainly some disadvantages over the standard 'laboratory' approach to experiments with human problem solving, particularly we do not have a direct control over our subjects. Nevertheless, we believe that the advantages significantly outweigh these disadvantages. We have been able to collect large number of data about

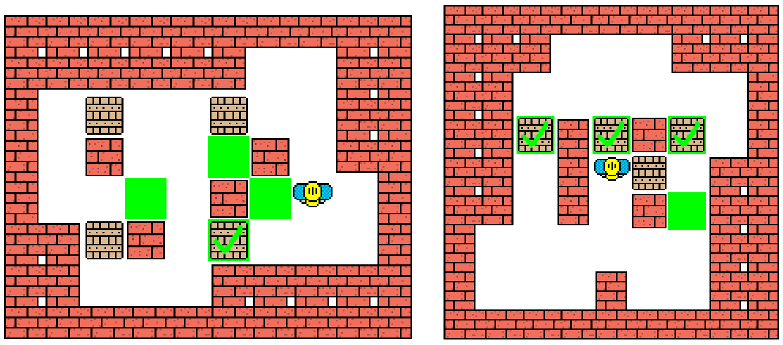


Figure 1: Example of two difficult Sokoban puzzles. The median solving time for the left problem (further denoted S071) is 43 minutes, for the right one (denoted S356) it is 49 minutes.

human problem solving activity: more then two thousand completed games. Almost three hundred people solved Sokoban on the portal and spent 785 hours with solving. This is much more than would be feasible with the classical laboratory approach. Moreover, the experimental setting is cheap and the data collection rather fast.

1.2 Sokoban

Sokoban is a logic puzzle which has simple rules and yet incorporates intricate dynamics and great complexity for both humans and computers. Example of the puzzle is in Figure 1. There is a simple maze with several boxes and one man. The goal is to get the boxes into the target squares by pushing one box at a time.

We have chosen Sokoban puzzle for several reasons. The first reason is interestingness. Sokoban has simple rules and intuitive visual setting; hence it is very simple to understand. Most people find it interesting and challenging and thus are willing to solve it voluntarily.

The second reason is availability of resources. There is a very large number of levels of the puzzle freely available on the Internet. These available levels span wide range of difficulty. Sokoban is also used by artificial intelligence community as a testbed for developing single agent search techniques; many techniques and results are described in research literature and state-of-the-art solvers are freely available (e.g., [12]).

The third reason is complexity. Despite the simplicity of the rules, Sokoban can be very challenging for both humans and computers; the puzzle also illustrates differences between humans and computers. There exist small instances that can be quickly solved by computer (using a trivial brute force algorithm) but take humans hours to solve. At the same time, there are also instances of the puzzle, which humans can solve but which are beyond capabilities of the state-of-the-art artificial intelligence solvers [12].

1.3 Problem Difficulty

We focused on the issue of problem difficulty. Previous research on the problem difficulty (e.g., [8, 11, 14, 15, 18]) was focused particularly on the following concepts:

- hill-climbing heuristic, which was studied for example for river crossing problems [11], Fifteen puzzle [18], and Water jug puzzle [4, 7],
- means-end analysis, which was proposed as a key concept in the "General Problem Solver" [16] and was studied for example for Tower of Hanoi puzzle,
- differences between comprehension of isomorphic problems, which focus on the difficulty of successor generation and were studied for example for Tower of Hanoi [14] and Chinese ring puzzle [14].

However, these concepts are not sufficient to explain differences in difficulty in Sokoban problems. In our experiment there are very similar problems with large difference in difficulty (more than 10-fold) – whereas the problems in Figure 1 took human on average nearly one hour, other problems were solved in within few minutes. Yet with respect to the above mentioned concepts the problems are nearly the same. Hill-climbing is not applicable for solving Sokoban problems (except very easy ones). Means-end analysis is applicable only in very limited sense and it is not clear how this concept could explain large differences in difficulty of different Sokoban problems. Differences between comprehension of isomorphic problems and successor generation also cannot be responsible for differences in difficulty, because all instances are stated in the same way.

1.4 Contributions

We describe a novel type of experiment with human problem solving using large scale data collection of human problem solving activity over the Internet. Using the collected data on Sokoban puzzle, we identify differences in problem difficulty that are not explained by previous research. To explain these differences, we develop an abstract computational model of human behaviour during search through a problem state space. We also describe a successful difficulty metric which is based on problem decomposition.

This technical report provides detailed description of the experiment and its results; it is meant as a rather complete documentation of the experiment for further references. Therefore, we also elaborate on techniques used for data analysis, particularly on our method of state space visualization. State space visualizations let us to study of a 'state space bottleneck' concept, for which we provide formalization based on network flows. Although the concept cannot be directly used as a difficulty metric, it provides an interesting insight into the issue of problem difficulty and could be useful for automated tutoring of problem solving skills.

2 Preliminaries and Methodology

In this section we describe the Sokoban problem, its state space, and methodology of our data collection on human problem solving.

2.1 Sokoban and Its State Space

Sokoban is a single player game that was created around 1980 in Japan[1]. Example of the puzzle is in Figure 1. There is a simple maze with several boxes and one man. The goal of the puzzle is to get the boxes into the target squares. The only allowed operation is a push by a man; man can push only one box. For more precise description of rules see, e.g., [1].

To analyze behavior of humans during problem solving, we explore their movement in the underlying problem state space. State space of the game is then formalized as a directed graph $G = (V, E)$ where V is the set of game states and E is the set of edges corresponding to a move of single box[2]. State of the game is thus given by a position of boxes in the maze and by area reachable by a man. We denote s_0 the vertex corresponding to the initial position of the game. In our discussion we consider only states reachable from the initial state.

There can be several states corresponding to a solved problem – all boxes have to be on given position in the final state, but there can be more final states due to the position of the man. Nevertheless, in nearly all cases there is just one final state; thus to simplify

[1]Sokoban is Japanese for "ware-house keeper"

[2]A naive formulation of a state space is to consider as a move each step of a man. This formulation does not add any important information to the analysis and leads to unnecessary large state spaces.

the discussion in the following we assume just one final state denoted s_f[3]. We say that a state s is "live" if there exists a path from s to s_f; otherwise we call the state s "dead".

2.2 Data Collection

To obtain data about human Sokoban solving, we used a web portal with a game simulator. Simulator was implemented in Javascript; all moves during the game were sent to the server, where they were stored into a database. We logged description of each move and time spent before the move.

For the experiment we used 35 problems, all of them had 4 boxes and similar size. To avoid bias from learning, we randomized order of problems for each player. Level were given non-explanatory names (fixed prefix + random looking three digit number). Moreover there were 4 sample training problems which players solved to gain the understanding of the problem; these training problems are not included in our analysis.

Participants were not paid and we did not have a direct control over them since the whole experiment run over the Internet. As a motivation to perform well there was a public results list – this is for most people sufficient motivation, and at the same time it is sufficiently weak so that there is not a tendency to cheat.

Summary statistics about the experiment are given in Table 1. In the next section we provide the analysis of data and show that they are sufficiently robust to be used for research purposes.

Table 1: Summary information about data collection.

participants who solved all 35	6
participants who solved at least 20	35
participants who solved at least 1 level	294
successful attempts to solve a problem	2071
all attempts to solve a problem	21511
total time spent solving	785 hours

[3]This simplification is only for sake of readability of the text. Implementations of all our techniques work correctly in the general case of multiple final states.

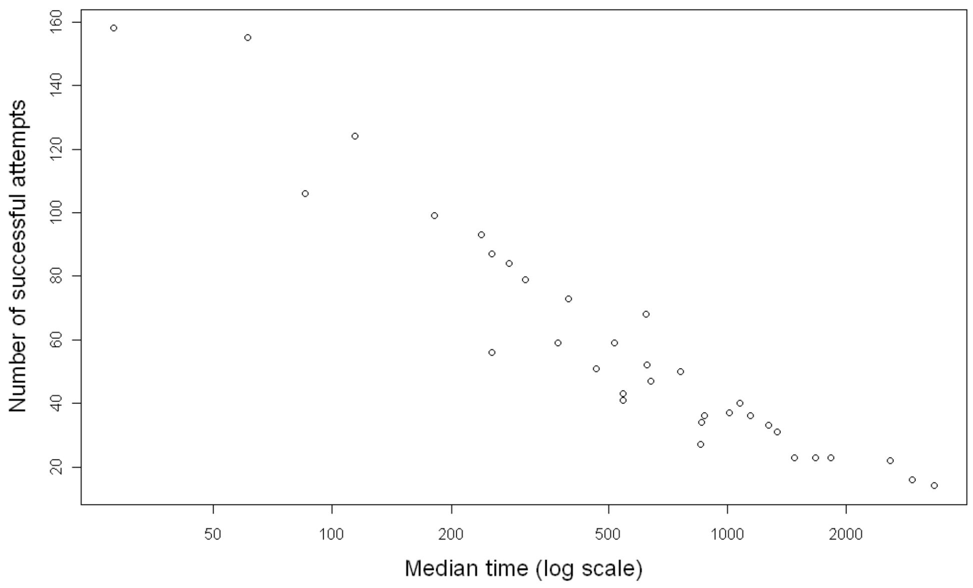


Figure 2: Correlation between median time to solve the problem and number of solvers who solved the problem.

3 Data Analysis

In this section we provide an analysis of the collected data. The data are publicly available and can be downloaded from:

http://www.fi.muni.cz/~xjarusek/vyzkum/sokoban/sokoban.tar

3.1 Problem Difficulty

Our aim is to explain and predict difficulty of individual problems. As the first step it is necessary to specify a fair and robust measure of difficulty. There are several natural measures of difficulty: time taken to solve a problem, number of moves necessary to solve a problem, number of solvers who successfully solved a problem. It turns out that all these measures are highly correlated, i.e., it seems plausible that any single of them sufficiently captures a concept of problem difficulty.

In the rest of the paper we use as a difficulty measure the median solving time of successful attempts. Figure 2 shows the relation between this measure and number of successful solvers ($r = -0.95$).

Figure 3 shows the distribution of solving time using the boxplot method; problems are sorted by the median time (our measure of difficulty). The solving time median varies from one minute to almost one hour. This result is interesting because we are

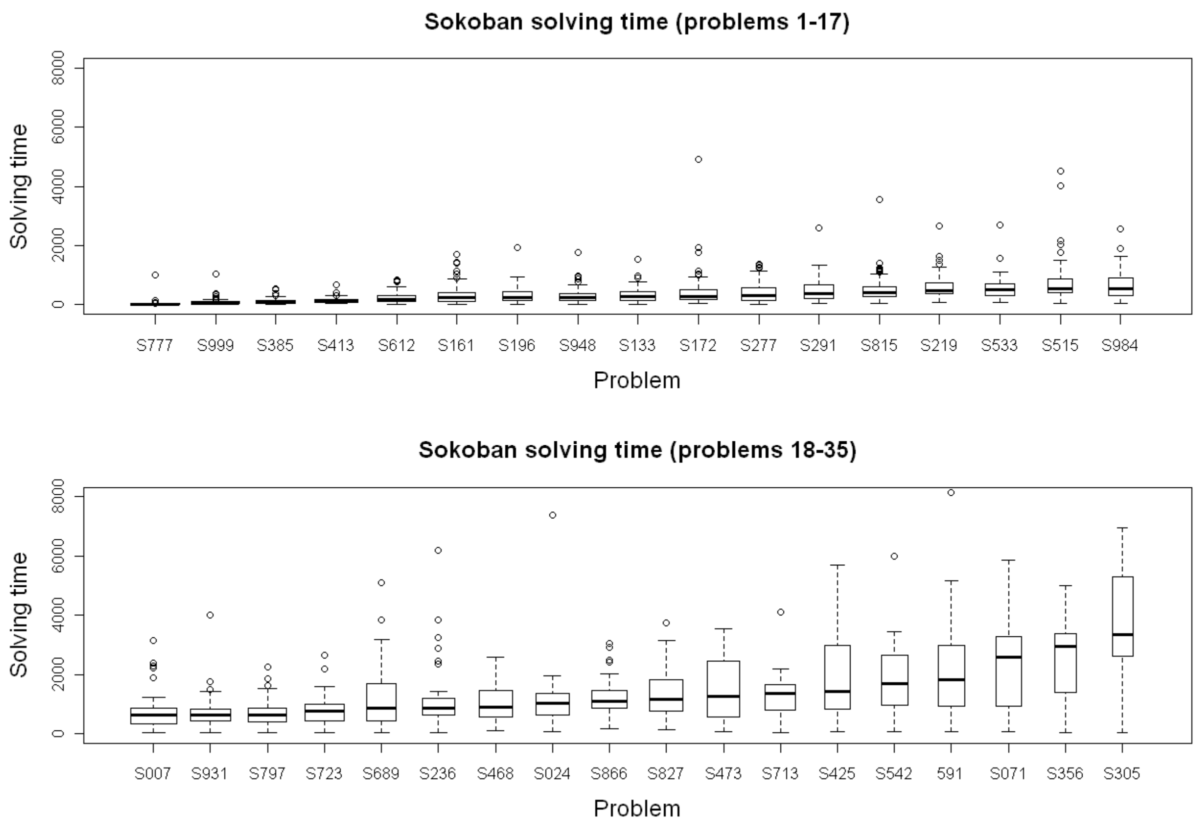


Figure 3: Boxplot of time to solve the Sokoban problems.

working with very similar Sokoban problems – all levels have exactly four boxes and the size and topology of the underlying maze is in all cases also very similar.

3.2 Analysis of Individual Moves

In our experiment we saved information about all moves performed by solvers (including time taken to make the move). Here we provide analysis of these individual moves. Based on results of this analysis we built our computational model of human problem solving behavior (Section 5).

Although 73.8% of game-configuration are dead (i.e. states from which is impossible to reach the goal), humans usually do not spent much time in dead states (14.5% on the average). They can relatively quickly discover that they are in bad configuration and restart the game.

Humans spent more time far from goal position (see Figure 4). Once humans get to one half of the distance between the start and goal state, they finish the problem rather quickly.

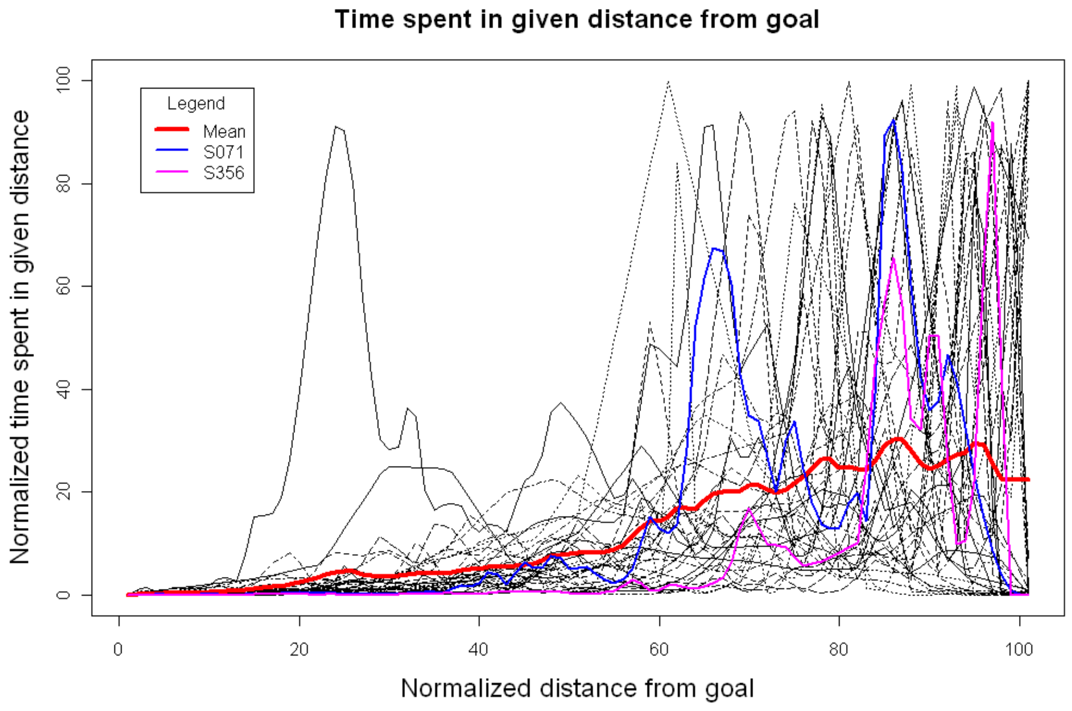


Figure 4: Normalized time spent in states with certain distance from goal. The bold line is a mean over all problems, two colored lines are problems from Figure 1.

To get better insight we visualized state spaces. But even for our rather small instances of Sokoban problems, whole state spaces are too big to be visualized directly, so it is necessary to prune the state space to obtain reasonable visualization. We prune state spaces in two ways. At first, we use only live states. This decreases the average number of states in the visualization from 3697 to 823; most important information is retained since humans spent most of the time in live states. At second, we cut long back-level edges[4]. According to our experience in most cases there are only few long back-level edges and their removal makes the visualizations much more comprehensible. Visualizations displayed in this paper cut off edges of length three or more, which we consider to be a reasonable compromise between loss of information and comprehension of visualization.

To visualize the pruned state space we use automated graph drawing[5]. Figure 5 shows examples of resulting visualizations. The size of each state is proportional to

[4]Back-level edges are edges which go to lower level with respect to breadth-first search.
[5]Specifically the tool Pajek [5], algorithm Kamada-Kawai [13].

average time spent by humans in the state, the thickness of each edges is proportional to the average number of times humans performed the given move.

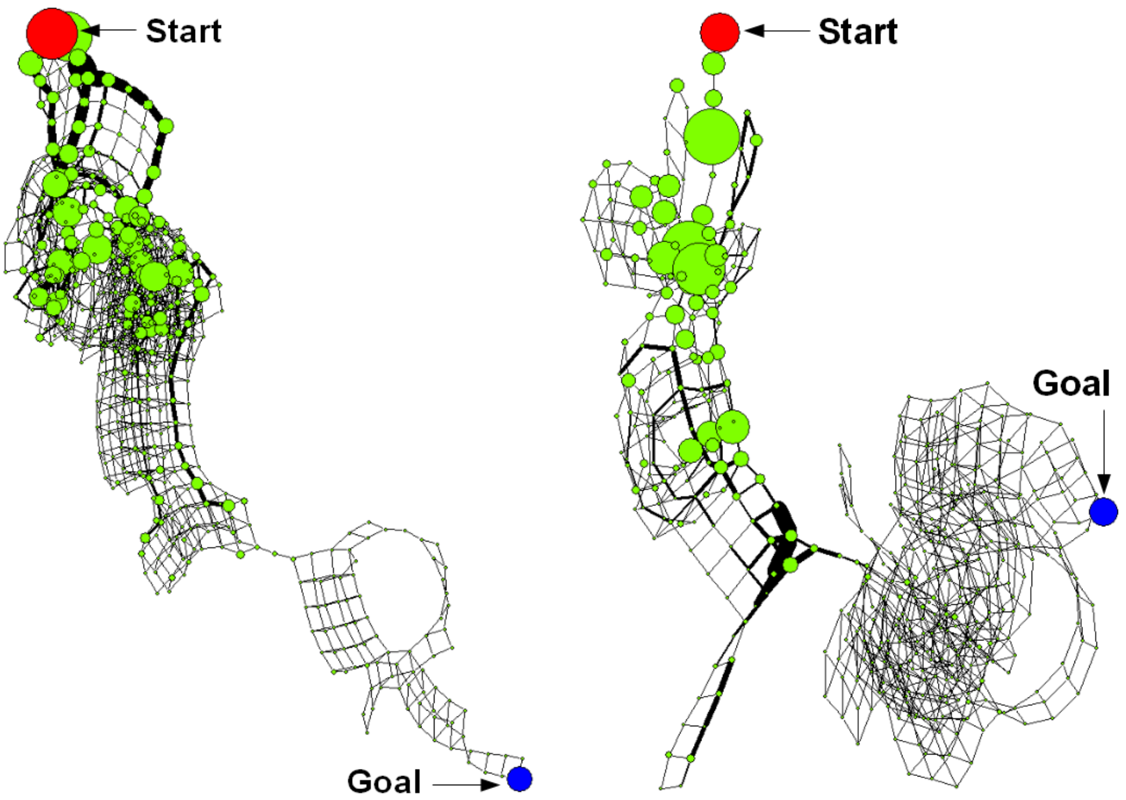


Figure 5: Examples of state space visualization of the problems given in Figure 1. Size of each vertex is proportional to average time spent by human solvers. Some edges are omitted from the visualization (see text).

4 State Space Bottleneck

Using visualizations of state spaces of difficult levels we identified a recurring feature – often there is a "bottleneck" (a narrow part in the state space) and people spent most of the time in states before the bottleneck (see Figure 5). Once people find the bottleneck, they usually quickly find a path to the final state.

Based on this observation we propose a formalization of the concept of a state space bottleneck using network flows with non-uniform price. The concept is based only on the structure of state space, i.e., it is not specific to Sokoban.

We are aware of only one related notion in the literature. Berlekamp et al. [6] briefly mention the notion of a 'narrow bridge' while discussing an abstract map of a state space of a sliding block puzzle Century. They attribute the high difficulty of the puzzle to existence of this ‘narrow bridge’ in the state space. They, however, do not provide any formalization of the concept.

4.1 Network Flows

A straightforward approach to formalizing the bottleneck concept is to employ graph connectivity notions, e.g., to find a minimum cut between initial and final state. This approach however has two disadvantages. At first, a bottleneck is not absolute measure, but rather a relative one – it is important to consider a “width” of the state space before the bottleneck, not just a “width” of the bottleneck. At second, connectivity measures are hard to compute. Therefore, we employ an approach based on network flows. Intuitively, we compute a maximum flow from the initial to final state and study in which states the flow accumulates.

Let G be a directed graph with two distinguished vertices, a source s and a sink t, and edge capacity function $c : V \times V \to \mathcal{N}$.A network flow is a function $f : V \times V \to \mathcal{R}$ with the following properties:

- capacity constraints: $\forall u, v : f(u, v) \leq c(u, v)$,
- skew symmetry $\forall u, v : f(u, v) = -f(v, u)$,
- flow constraint: $\forall v : v = s \vee v = t \vee \Sigma_{(u,v)\in E} f(u, v) = \Sigma_{(v,w)\in E} f(v, w)$.

A maximal flow can be computed by Ford-Fulkerson algorithm [9]. The algorithm chooses an augmenting path from the source to the sink and increases the flow along this path. This process is repeated until there is no path that can be augmented. This basic algorithm is not suitable for finding a bottleneck, because it pumps the shortest path through the graph to the capacity of the minimal cut (see Figure 6). We need to make the flow more uniform. To achieve this goal we associate price with each edges and find a ‘minimum cost maximal network flow’ [2]. Pricing function p gives every edge (u, v) its real valued price depending on the flow $f(u, v)$. We consider only simple polynomial price function of the current flow $f(u, v)$, particularly a quadratic price.

To find a minimum cost maximal network flow, we can use an extension of the Ford-Fulkerson algorithm [2]. When choosing an augmenting path, we choose the cheapest one with respect to price function (this can be done efficiently by Dijkstra’s algorithm).

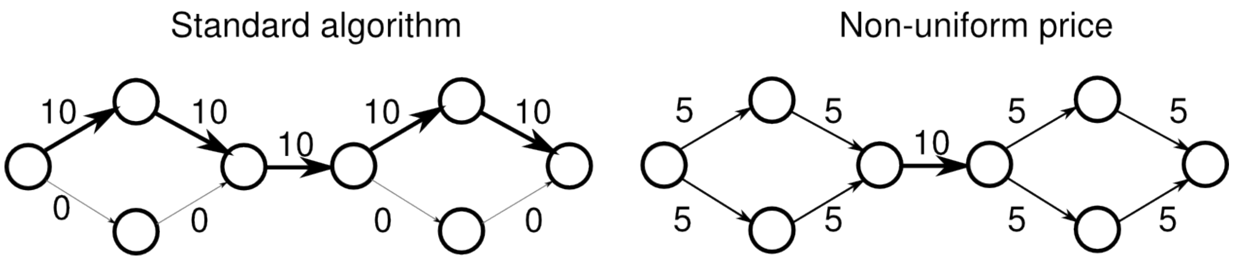


Figure 6: Example of a network and maximum flow (all edges have capacity 10). Flow computed by standard Ford Fulkenson algorithm does not identify the bottleneck; the flow computed with non-uniform prices splits the flow among more edges and thus finds the bottleneck.

The resulting flow spreads the flow in the wide part of state space and keeps a large flow through the bottleneck (see Figure 6).

4.2 Bottleneck Coefficient

Even the network flow computed with respect to non-uniform price does not directly identify bottleneck states. Particularly, states close to the initial and final state typically have a large flow, but they should not be considered as bottleneck states. We need to further 'recalibrate' the results. For each state we compute how much the flow 'spreads' before and after this state. Let f be the minimum cost maximal flow. We define flow spread before vertex v (denoted $s_b(v)$) as a maximum of $\min_{u \in p} f(u)$ over all paths p from the initial state to v, analogically flow spread after vertex v (denoted $s_a(v)$) is a maximum of $\min_{u \in p} f(u)$ over all paths from v to the final state. Bottleneck coefficient of vertex v is then defined as $b(v) = f(v)/(s_b(v)^2 + s_a(v)^2)$.

Bottleneck coefficient can be computed efficiently since s_b and s_a can be computed easily using a simple dynamic programming algorithm. Figure 7 provides example of resulting bottleneck coefficients for two state spaces. We see that the coefficient clearly identifies what we would naturally call a bottleneck.

4.3 Possible Applications

Our study of bottleneck was motivated by visual analysis of state spaces of difficult Sokoban problems. Can we use bottleneck coefficients as a difficulty rating metric? Unfortunately, it does not seem so. A straightforward metric based on bottleneck coefficients (maximum over all coefficients in the state space) gives poor correlation with

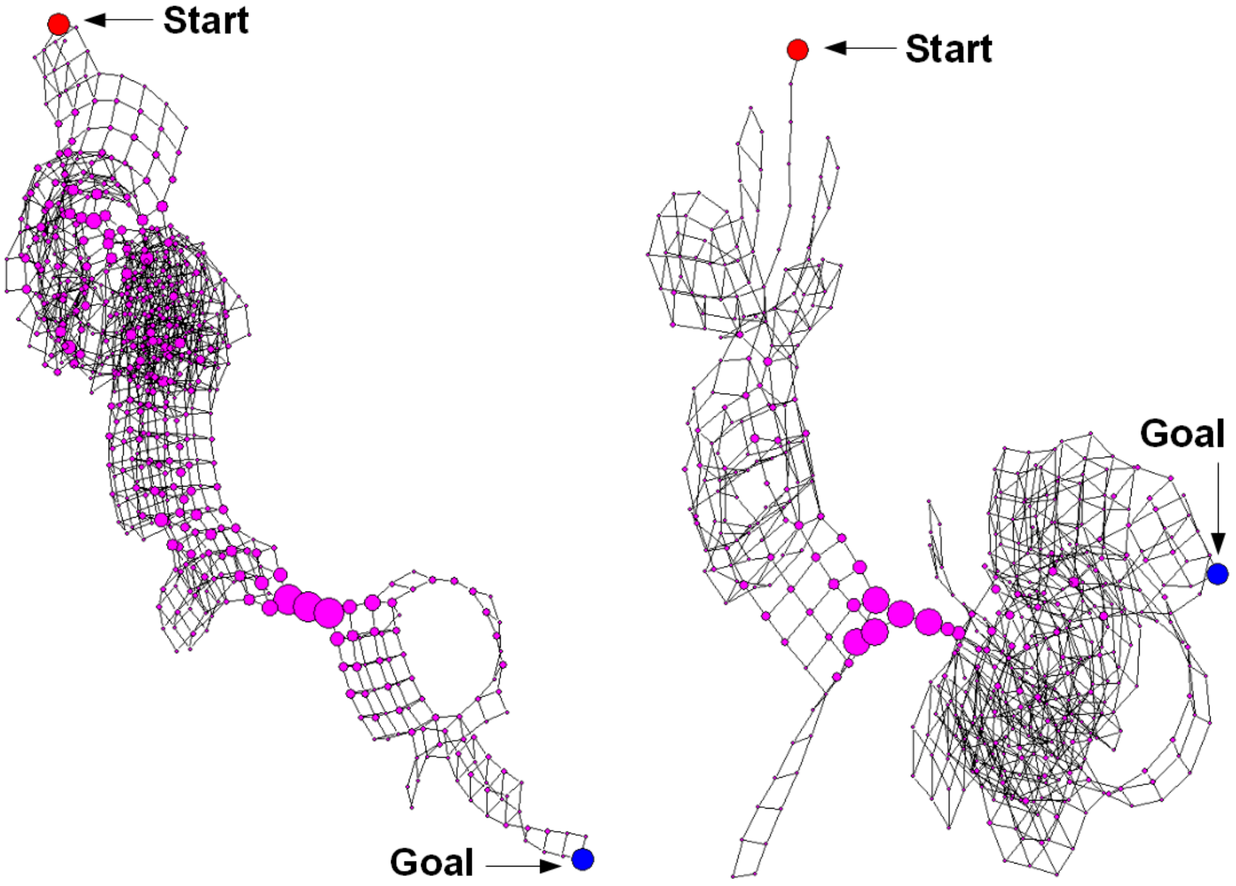


Figure 7: Bottleneck coefficients. State space for two sample problems from Figure 1, size of each vertex v corresponds to bottleneck coefficient $b(v)$.

problem difficulty. Some problems are difficult even though the set of live states is rather small and thus the bottleneck is not very dominant (even though it exists). Some problems have “strong” bottleneck, but are not very difficult, since the structure of the state space and the problem makes it easy to find the bottleneck. So to serve as a difficulty metric it would be necessary to combine the bottleneck coefficient with other information about the structure of the state space and it is not clear how to do it.

Nevertheless, the concept of bottleneck could be useful for example for understanding human behavior or for tutoring humans during problem solving. If a human cannot solve a problem, bottleneck states can serve as useful hints, since they provide natural decomposition of the problem. By their nature they even provide some kind of insight into the problem structure. Their explicit identification could help humans to learn to better understand problems. These issues require further study.

Note that the concept depends only on the state space and is completely independent of the particular problem. Therefore it should be applicable to other problems as well. For example, the much studied Tower of Hanoi problem does have a very strong bottleneck in its state space; this bottleneck state corresponds to a natural problem decomposition identified by means-end analysis.

5 Computational Model of Human Solver

In this section we describe a simple computational model of human Sokoban solving. Our goal is to replicate only the human behavior in terms of state space traversal characteristics. We do not try to model the actual human cognitive processes while solving the problem, i.e., this is cognitive engineering model rather than cognitive science model [10]. Our model is very abstract and is based only on information about underlying problem state space, i.e., the model is not specific for Sokoban. In the next section we use the model for predicting problem difficulty.

5.1 Basic Principle

Our model is based on the analysis of human behaviour as discussed in Section 3.2. At the beginning humans explore the state space rather randomly, later, as they get closer to the solution, they move more straightforwardly to the goal. Since humans spent most time at live states, the basic model works only with these states and completely avoids dead states.

The model starts at the initial state and then repeatedly selects a successor state. This selection is in the basic model local and very simple – it is a combination of two tendencies:

- random walk – select a random successor,
- optimal walk – select a successor which is closer to the goal state.

Human decisions are usually neither completely random, nor completely optimal. Nevertheless, the model assumes that a weighted combination of these two tendencies can provide a reasonable fit of human behaviour.

5.2 Model Formalization

The general principle of our model is the following. In each step the model considers all successors s' of the current state s. Each successor s' is assigned a value $\mathit{score}(s')$, the sum of all *score* values is denoted *SumScore*. The model moves to a successor which is selected randomly according to a probabilistic distribution:

$$P(s') = \mathit{score}(s')/\mathit{SumScore}$$

This general model is specified by a selection of a *score* function. In this report we evaluate the basic version of the model which uses a simple function based on distance $d(s)$ of a state s from the goal state. The function is defined as follows (B is a single parameter of the model – 'optimality bonus'):

$$\mathit{score}(s') = \begin{cases} 0 & d(s') = \infty \\ d(s) & d(s') \neq \infty, d(s') \geq d(s) \\ d(s) + B & d(s') < d(s) \end{cases}$$

Let us discuss the intuition and rationale behind this formula. The first case means that dead states have zero probability of being visited, i.e., the model visits only live states . The third case means that successors that lead to the solution get an 'optimality bonus', i.e., they have higher chance of being selected. The use of distance from the goal in the formula has the consequence that the relative advantage of 'bonus' increases as the model gets closer to the goal, i.e., the model behaves less randomly when it is close to the goal (as do humans).

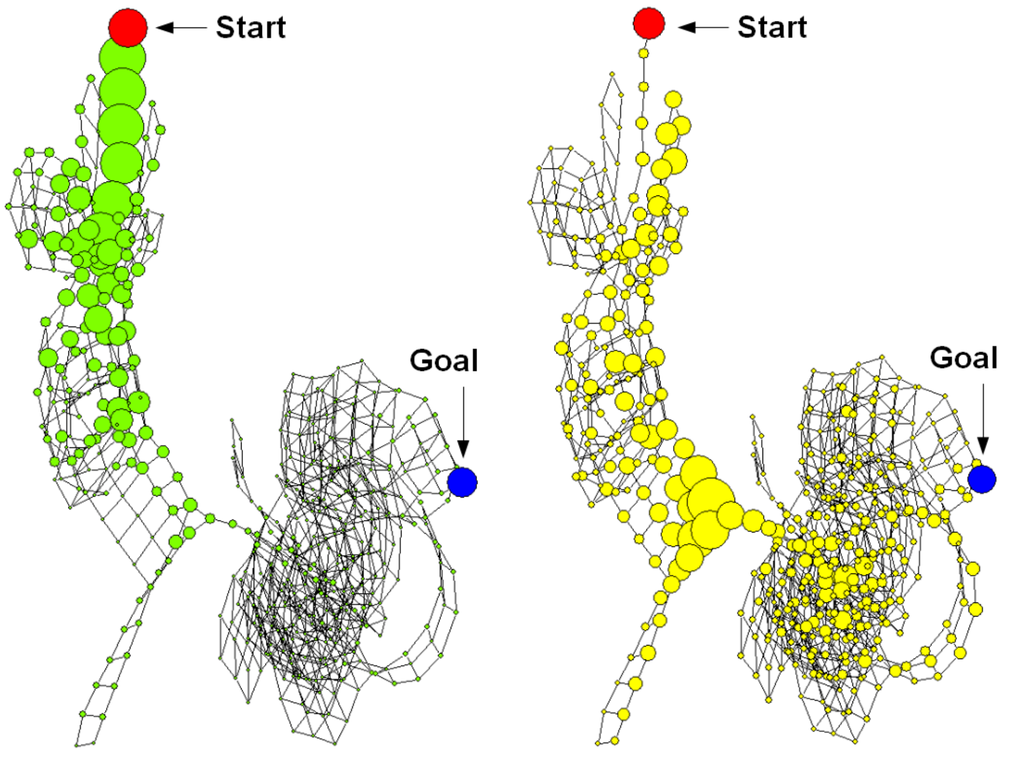


Figure 8: Comparison of human visits (green) and model visits (yellow) for the problem S356.

If $B = 0$ than the model behaves as a pure random walk (within live states). As B increases the behaviour of the model converges to the optimal path. Hence by tuning the parameter B the model captures continuous spectrum of behaviour between randomness and optimality.

Figure 8 gives an example of a comparison of human and model state space traversal.

5.3 Possible Extensions

In this work we discuss and evaluate just the simplest reasonable version of the model. Nevertheless, the model can be further extended in several ways. The extensions are quite natural and can be done simply by extending the scoring function:

- Hill climbing heuristic (specific for the particular problem, i.e., Sokoban). For Sokoban the natural heuristic is the total distance of boxes from goal positions.
- Use of memory (loop avoidance heuristic). The model would remember states that were already visited and in the scoring function would prefer unvisited states.
- Exploration of dead states (with lower, but non-zero, probability than live states).

- Penalization of long back edges. Humans can recognize not just moves which lead to dead states, but also moves which lead "backwards".

Each of these extensions incorporates at least one additional parameter into the model. With the size of our testing data (35 problems) it could be misleading to evaluate versions of the model with more parameters due to the potential overfitting of data [19].

6 Difficulty Rating Metrics

In this section we study metrics for assessing difficulty of Sokoban problems. We discuss several types of metrics and then we provide evaluation using the data collected in our experiment (as described in Section 3).

6.1 Metrics Based on State Space

The most straightforward approach, which was also initially proposed by Newell and Simon [21], is to use properties of state spaces as a metric of problem difficulty. We evaluate the following ones:

- number of states in the state space,
- number of live states in the state space,
- average "live" degree (number of live successors) of live states.

6.2 Metrics Based on Shortest Paths

The most intuitive difficulty metric is the 'number of steps necessary to reach a solution', i.e., the length of the shortest path from initial to goal state in the state space. Other metrics can be obtained as variations on this basic principle.

One of the concepts which was intensively studied in previous research on problem solving is the hill-climbing heuristics [4, 7, 18]. This concept can be quite directly applied as a difficulty metric. The straightforward hill-climbing heuristic for Sokoban is to minimize the total distance of boxes from their goal positions. Given this heuristic, we can define the metric 'counterintuitive moves' as a number of steps on the shortest path which go against this hill-climbing heuristic.

Similar metric is based on the number of changes of connected areas in the problem maze. Area is a part of the maze which can be reached by a man without pushing a box.

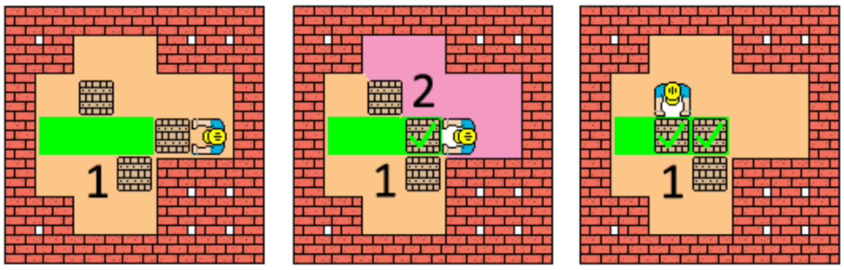


Figure 9: Illustration of the ‘area change’ metric. The example shows three consequential states of a Sokoban problem. The number of areas in states is one, two, and one; thus the ‘areas change’ metric for this sequence is computed as |1-2| + |2-1| = 2.

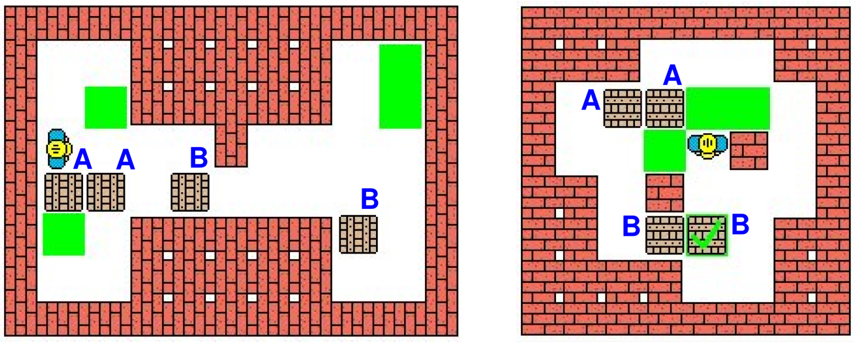
We define a metric ‘areas change’ as a sum of sizes of differences of areas counts on the shortest path (see Figure 9). The metric assumes that humans are less willing to make moves which change areas.

Another intuitively important concept in problem solving is problem decomposition. Humans are not good at systematic search, but they are good at tasks such as abstraction, pattern recognition, and problem decomposition. If a problem can be decomposed into several subproblems it is usually much simpler (for humans) than a same type of problem which is highly interwoven and indecomposable (see example in Figure 10). The concept of problem decomposition is however more difficult to grasp than the hill-climbing heuristic.

We propose a way to formalize problem decomposition for a Sokoban puzzle. A natural unit of ‘composition’ is a single box. Thus we can consider decomposition of a problem into single boxes and than count as a single move any series of box pushes. We can also generalize this idea and decompose the problem into two pairs of boxes[6] and than count as a single move any series of box pushes within the group.

Let D be a division of n boxes into several groups (at most n), in our case $n = 4$ and we denote the division by 4 letter string; e.g., ’ABAB’ is a division in which the first and the third box are in group A, the second and the fourth box are in group B; ’ABCD’ is a division in which each box is in a separate group. Each edge in the state space is labelled by identification of the group to which the moved box belongs. Let p be a path in a state space (a sequence of valid moves). We are interested in the number of label alternations $a(p, D)$ along the path.

[6]Remember that all our problems contain 4 boxes.



problem	decomposition			
	ABCD	**AABB**	ABAB	ABBA
left problem	10	**2**	6	5
right problem	14	**7**	12	10

Figure 10: Example of two Sokoban puzzle; the first one can be easily decomposed into two subproblems and is thus easy (median solving time 3:02 minutes), the second one is rather indecomposable and thus very difficult (median solving time 53:49 minutes). The table gives the number of 'steps' for different decomposition (see text). The bold column corresponds to the decomposition provided in the figure.

Optimal solution of the problem with respect to a division D (denoted $s(D)$) is the minimum $a(p, D)$ along all paths from the initial to the goal state. This optimal solution can be computed by the Dijkstra algorithm over an augmented state space – vertices are tuples (s, g) where s is a state in the state space and g is an identification of a group and edges have weights 0 or 1. Figure 10 gives results for different decompositions of the two provided examples.

For our evaluation we use two metrics based on these concepts. At first, we use the 'box change' metric which is based on the division 'ABCD', i.e., each box is a single group. At second, we use the '2-decomposition' metric, which is the minimum number of steps over all possible division into two groups (division 'AABB', 'ABAB', 'ABBA'). We also tried other types decompositions (e.g., 3-1 decomposition such as 'AAAB'), but the results were similar to 2-decomposition and we do not report them explicitly.

6.3 Metrics Based on the Computational Model

The model described in Section 5 can be easily used as a basis for a difficulty metric. We just run the model repeatedly and take the number of steps it took the model to reaching the goal state. The average number of steps is used as a difficulty metric.

6.4 Evaluation and Discussion

Based on results from Section 3, we take as a 'real' measure of problem difficulty a median solving time of human solvers. Figure 11 provides scatter plots showing relation among several of the described metrics and the real difficulty.

To quantify the quality of a metric we use correlation coefficients. Except for the standard Pearson correlation coefficient, we also measure Spearman correlation coefficient, which gives the correlation with respect to ordering of values – for practical application of difficulty metrics the ordering is often more important than absolute values.

Table 2 provides overview of all results. The results shown for metric based on model correspond to the optimal choice of the bonus parameter B. Figure 12 provides sensitivity analysis of the metric based on the computational model with respect to the parameter – Spearman coefficient is quite stable, but Pearson coefficient is not.

Table 2: Correlation coefficients for different difficulty metrics, results given in bold are statistically significant ($\alpha = 0.05$).

type	metric	Pearson	Spearman
state space	size	-0.11	-0.07
	number of live states	-0.17	-0.15
	average "live" degree	-0.24	-0.36
shortest paths	shortest path	0.30	**0.47**
	counterintuitive moves	**0.52**	**0.69**
	area change	0.24	**0.35**
problem decomposition	box change	**0.51**	**0.74**
	2-decomposition	**0.63**	**0.82**
model	average number of steps, $B = 25$	**0.76**	**0.66**

Metrics based on state space do not work at all (no statistically significant correlation). The intuitively plausible metric based on length of the solution is better and

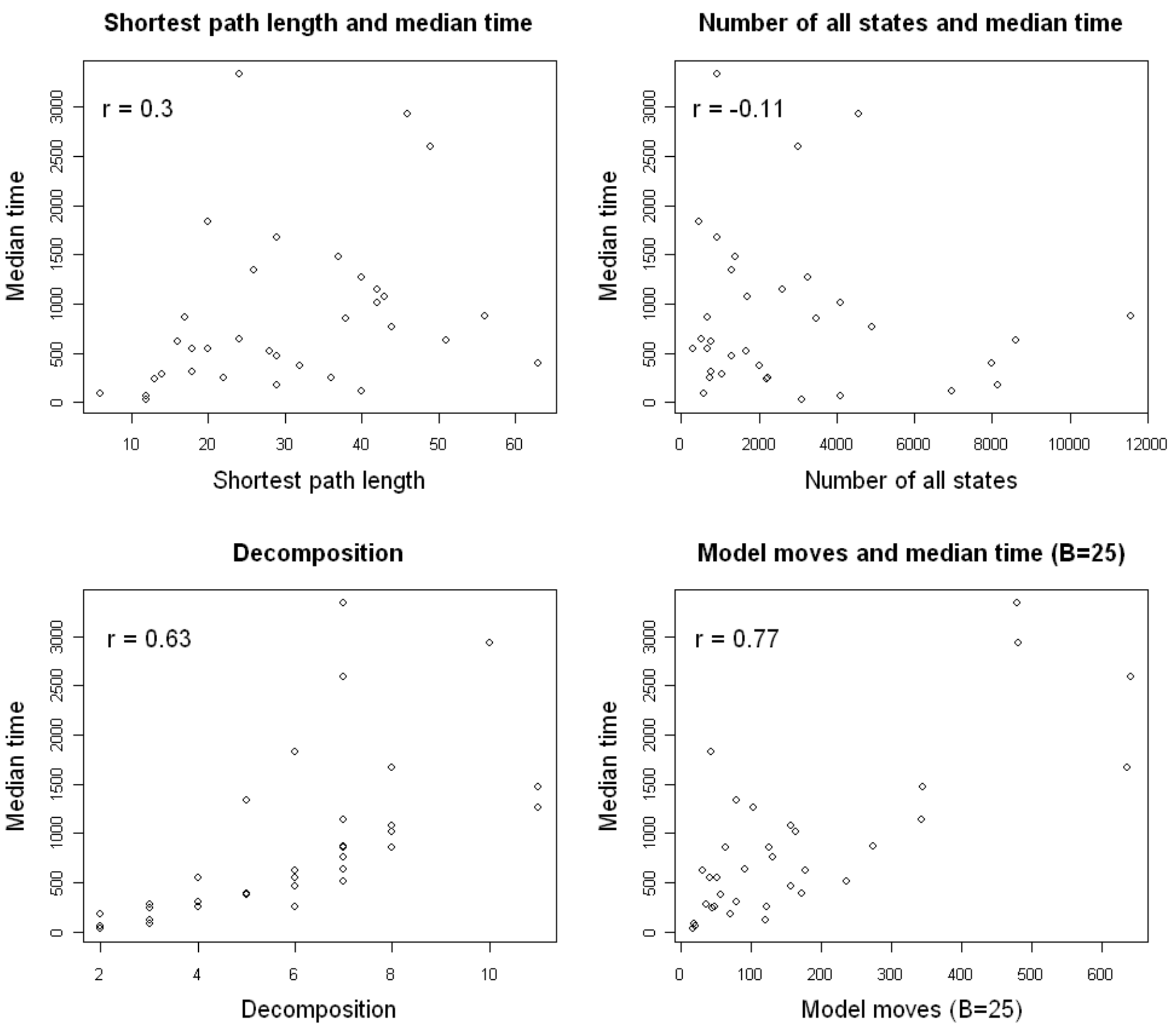


Figure 11: Scatter plots for 4 different difficulty metrics.

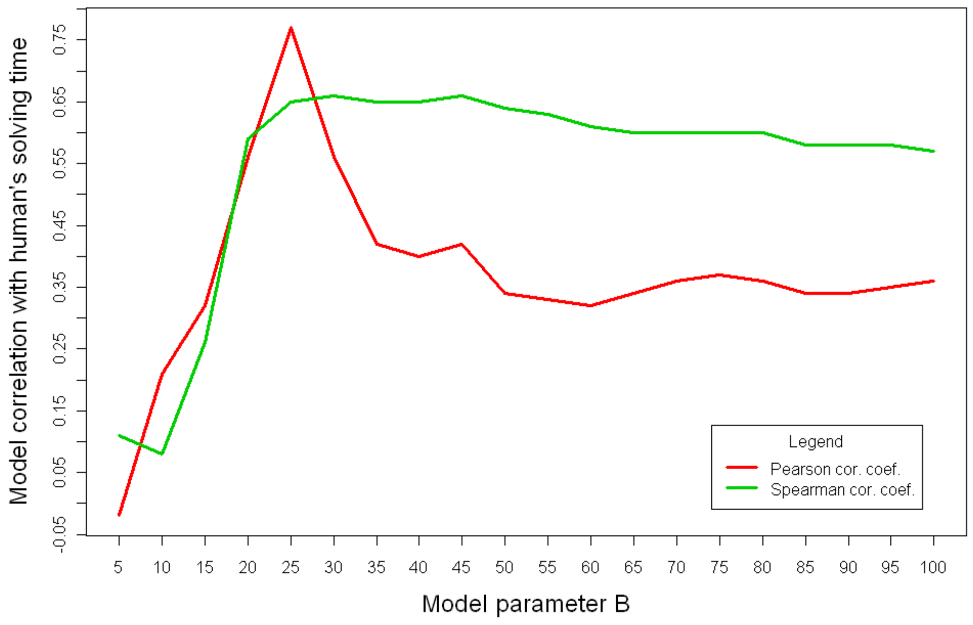


Figure 12: Quality of predictions of problem difficulty by metric based on the computational model depending on the model parameter B (given as correlation between human median time and average number of model steps for our levels).

further improvement is brought by Sokoban specific extension of shortest paths (counterintuitive moves, area change metrics). The best results are obtained by the metric based on problem decompositions and by the metric based on computational model. We believe that the results of the computational model can be further improved by extensions discussed in Section 5.3, but at the moment we do not have enough data to fairly evaluate a model with several free parameters.

7 Conclusions and Future Work

In this paper we describe a case study in human problem solving. For a particular problem – a Sokoban puzzle – we describe a method of collecting data on human problem solving, an analysis of the collected data, an abstract computational model of human problem solving activity, and evaluation of several difficulty metrics.

In our analysis we focus particularly on the issue of problem difficulty – differences in difficulty of studied problems are large (more than 10-fold) even though the problems are very similar. These differences are not explained by previous research. Using our computational model and the concept of problem decomposition we are able to partially explain these differences. We also propose a concept of state space bottleneck and provide its formalization using network flows with a non-uniform price. Although the concept is not directly applicable for explaining differences in problem difficulty, we believe that it provides an interesting insight into the nature of difficult problems and that it may be useful for automated tutoring.

There are many direction for future research:

- evaluation of the computational model on other well-defined problems, particularly on other "move" puzzles (e.g., Tower of Hanoi, Water jug problem, sliding block puzzles),
- refinement of the concept of state space bottleneck,
- application and evaluation of the decomposition metric to larger Sokoban problems and to other problems,
- more detailed study of data on human problem solving, particularly study of differences among individual solvers,
- application of results in an intelligent tutoring system [3] for training of problem solving skills.

Acknowledgement

We thank Ondřej Bouda for assistance with the web experiment and Aymeric du Peloux, David W. Skinner, M. Hiroshi, and Martí Homs Caussa for creating some of the Sokoban problems which were used in our experiments.

References

[1] Sokoban wiki. `http://www.sokobano.de/wiki`.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[3] J.R. Anderson, C.F. Boyle, and B.J. Reiser. Intelligent tutoring systems. *Science*, 228(4698):456–462, 1985.

[4] M.E. Atwood and P.G. Polson. Further explorations with a process model for water jug problems. *Memory & Cognition*, 8(2):182–192, 1980.

[5] V. Batagelj and A. Mrvar. Pajek – analysis and visualization of large networks. In *Graph Drawing*, volume 2265 of *LNCS*, pages 8–11. Springer, 2002.

[6] E.R. Berlekamp, J.H. Conway, and R.K. Guy. *Winning ways for your mathematical plays*. AK Peters Ltd, 2003.

[7] H.P. Carder, S.J. Handley, and T.J. Perfect. Counterintuitive and alternative moves choice in the Water Jug task. *Brain and Cognition*, 66(1):11–20, 2008.

[8] M. Dry, M.D. Lee, D. Vickers, and P. Hughes. Human performance on visually presented traveling salesperson problems with varying numbers of nodes. *Journal of Problem Solving*, 1(1):20, 2006.

[9] L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[10] W. D. Gray. *The Cambridge Handbook of Computational Psychology*, chapter Cognitive Modeling for Cognitive Engineering, pages 565–588. Cambridge University Press, 2008.

[11] J.G. Greeno. Hobbits and orcs: Acquisition of a sequential concept. *Cognitive Psychology*, 6(2):270–292, 1974.

[12] A. Junghanns. *Pushing the limits: New developments in single-agent search*. PhD thesis, University of Alberta, Department of Computing Science, 1999.

[13] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[14] K. Kotovsky, J.R. Hayes, and H.A. Simon. Why are some problems hard? Evidence from tower of Hanoi. *Cognitive psychology*, 17(2):248–294, 1985.

[15] K. Kotovsky and H.A. Simon. What Makes Some Problems Really Hard: Explorations in the Problem Space of Difficulty. *Cognitive Psychology*, 22(2):143–83, 1990.

[16] A. Newell and H.A. Simon. GPS, a program that simulates human thought. *Computers and thought*, pages 279–293, 1963.

[17] Z. Pizlo. Human Problem Solving in 2006. *The Journal of Problem Solving*, 1(2):3, 2007.

[18] Z. Pizlo and Z. Li. Solving combinatorial problems: The 15-puzzle. *Memory and Cognition*, 33(6):1069, 2005.

[19] S. Roberts and H. Pashler. How persuasive is a good fit? A comment on theory testing. *Psychological Review*, 107(2):358–367, 2000.

[20] J. Schaeffer and H.J. Van den Herik. Games, computers, and artificial intelligence. *Artificial Intelligence*, 134(1-2):1–8, 2002.

[21] H.A. Simon and A. Newell. *Human problem solving*. Prentice Hall, 1972.

[22] L.G. Terveen. Overview of human-computer collaboration. *Knowledge Based Systems*, 8(2):67–81, 1995.

[23] R.A. Wilson and F.C. Keil. *The MIT encyclopedia of the cognitive sciences*. MIT Press, 1999.