UNIVERSITÄT BASEL

Phase Transitions in the Solvability of Sokoban

Bachelor's Thesis

Natural Science Faculty of the University of Basel
Department of Computer Science
Artificial Intelligence
http://ai.cs.unibas.ch

Examiner: Prof. Dr. Malte Helmert
Supervisors: Dr. Martin Wehrle, Jendrik Seipp

Beat Hänger
beat.haenger@stud.unibas.ch
03-066-701

18-12-2013

UNI
BASEL

Acknowledgments

I thank my supervisors, Martin Wehrle and Jendrik Seipp for their guidance. I cordially thank my family for their support during this time. I especially thank my brother, Urs Hänger, who designed the level visualisations and his wife Brigitte Hänger who provided important insights into statistical analysis.

Abstract

Phase Transitions in the Solvability of Sokoban

Sokoban is a computer game where each level consists of a two-dimensional grid of fields. There are walls as obstacles, moveable boxes and goal fields. The player controls the warehouse worker (Sokoban in Japanese) to push the boxes to the goal fields. The problem is very complex and that is why Sokoban has become a domain in planning.
Phase transitions mark a sudden change in solvability when traversing through the problem space. They occur in the region of hard instances and have been found for many domains. In this thesis we investigate phase transitions in the Sokoban puzzle. For our investigation we generate and evaluate random instances. We declare the defining parameters for Sokoban and measure their influence on the solvability. We show that phase transitions in the solvability of Sokoban can be found and their occurrence is measured. We attempt to unify the parameters of Sokoban to get a prediction on the solvability and hardness of specific instances.

Table of Contents

1 Introduction

Sokoban is a computer game in which the agent moves boxes to their destination on a two-dimensional grid. The boxes can only be pushed, not pulled, and the task is further complicated by walls that act as obstacles. Because getting boxes stuck happens so quickly, solving a level is all the more rewarding. Sokoban has become a domain for automated planning for two reasons: it is difficult and has a real-life component.
Humans are excellent at recognising patterns and applying intuitive strategies. Computers often have troubles solving Sokoban levels because they can literally get lost in the overwhelming amount of possible solution approaches. Problems that occur in real-life often have many factors and are embedded in a context of interdependent subproblems. Yet, simplifications can be carried over to the world of automated planning and most standard domains are based on real world problems. Sokoban has such similarities to real world problems.
When working on a certain problem domain the goal is to get specific knowledge. This can be done by analysing the theoretical concepts behind the domain, thus reducing and abstracting the problem or by examining statistical data of problem sets and searches.
A unique way to obtain domain specific knowledge is to investigate *phase transitions*. Each domain has its defining parameters. After solving randomly generated sets with varying parameters, the solvability and the resources required for the search are analysed.
Phase transitions are regions in the traversed problem space for which sudden changes in required resources and solvability occur. Problems go from being under-constrained, leaving many liberties to the solver to over-constrained, leaving very little liberties to the solver. Outside the phase transition problems are either predominantly solvable or unsolvable, while the phase transition marks the region of abrupt change. Within this region lie the problems that are the most resource intensive to solve.
This gives a direct link between the problem's properties and its supposed difficulty. Knowing where the hard problems are is interesting because search algorithms and heuristics can be better optimised and evaluated with hard instances.
Phase transitions were found for many problems, including some that share Sokoban's complexity class, therefore they can be assumed to also exist for Sokoban. The objective of this thesis is to investigate phase transitions for the Sokoban puzzle and find descriptive parameters to predict the hardness of randomly generated Sokoban levels.

The further structure of this thesis is as follows: Chapter 2 describes the needed concepts and imparts basic knowledge about search problems, phase transitions and the Sokoban problem. Chapter 3 explains which decisions were made when investigating phase transitions and the reasoning behind them. The results from the conducted experiments are presented, visualised and discussed in Chapter 4. The conclusions and gained insights from this work are outlined in Chapter 5. It also contains thoughts on possible future work.

2 Preliminaries

Section 2.1 first explains search and planning problems and the corresponding terms. Secondly, Section 2.2 discusses phase transitions, their origin, meaning and importance in planning. A special regard is given to the k-SAT problem as the prime example for finding such phase transitions. Section 2.3 presents Sokoban as the problem for which phase transitions are investigated. Sokoban's history, properties and complexity are described, a comparison to k-SAT is made and the used problem language (PDDL) introduced.

2.1 Planning Problems

Planning problems play a central role in computer science. They are at the heart of artificial intelligence and many real life problems can be represented by them. Such a real life problem is for example the optimal distribution of cargo.
A standard way to describe planning problems is by an *initial state* with its entities and their conditions, a *set of actions* by which the world can be altered and a set of final, desired states with their properties, called the *goal states*. The task is to discover a sequence of actions that transforms the initial state to a goal state. A legal sequence that successfully achieves that is called a *plan*.
Actions can have different costs and finding one of the plans with the lowest cost is the so-called *optimisation problem*. When declaring a certain plan as optimal, all other plans must be justifiably deemed more or just as costly and that is what makes optimal planning more difficult than planning without optimisation. To each search problem there is a corresponding *decision problem* that can be formulated as: Does any plan exist? A problem for which this question can be answered with yes, is solvable. The ratio of solvable versus unsolvable problems is called *solvability* and it increases with more solvable problems.
Programs that solve planning problems are called *planners*. A broad class of planners is based on search. When conducting such a search, it is important to get an idea of how good a regarded state is. Without such a mechanism each state would be seen as equally promising and following unfavourable paths of actions would be equally likely as pursuing favourable paths. *Heuristics* are functions that evaluate states in regard to their distance to a goal state. Such functions must master the trade-off between fast execution and reliable result,

between accuracy of estimation and practicality. Therefore, heuristics play a fundamental role in automated planning.

To improve searches two approaches can be differentiated: the first is to find algorithms and heuristics that work well for many different domains and have a good overall performance; the second is to use domain specific knowledge to optimise the search for certain problems or classes of problems. Since such specific methods can then be generalised, new ideas for one domain can be gained by looking at successful techniques for other domains.

2.2 Phase Transitions

Phase transitions in artificial intelligence and planning are derived from phase transitions in physics. For example, while a small and gradual change of temperature results in a small and gradual change in behaviour for most of the temperature range, there are regions where a small change in temperature results in a sudden and dramatic change in behaviour. These are the regions where materials for example change from solid to liquid and from liquid to gaseous. Although different materials have their phase transitions at very different temperatures because of their distinct properties, a region of phase transition can be found for any material [1].

There are other areas in computer science where phase transitions have been investigated, for example in classification and pattern matching [2]. This thesis is concerned with the phase transitions that occur in the solvability of a specific problem.

Analogously to physics these phase transitions happen abruptly when traversing steadily through the defining parameters of a problem. The two phases when observing solvability are solvable and unsolvable. As in physics, the idea is that such transitions occur in any problem of certain classes.

2.2.1 Phase Transitions in k-SAT

The most basic kind of variable has only two possible values: true or false. Such variables are called *Booleans*. In propositional logic, Booleans can be joined together to form logical expressions. A disjunction, symbolised by $\vee$, is overall true when at least one of the Booleans is true. The sign for conjunction, the logical "and", is $\wedge$. The negation, $\neg$, inverts the value of an expression or Boolean to the opposite.

For the propositional or Boolean satisfiability problem (SAT) a logical expression is investigated for a configuration that makes the expression true. The expressions consist of clauses that contain Booleans joined by disjunction and the clauses among themselves are joined by conjunction. The Booleans can either be negated in the formula or not. For instance, a SAT formula could look like this:

$$(A \vee \neg B \vee C) \wedge (\neg A \vee D \vee B)$$

A satisfying configuration for this problem is that all variables are true. This problem is *under-constrained* which means that there are many different solutions and most variables can be freely assigned to either true or false without jeopardising the solution. In *over-constrained* problems, certain variables appear in many different clauses, making it impossible to find a satisfying configuration and most often rendering the problems unsolvable. In this example the number of Booleans per clause is three, making it a 3-SAT problem.

However, the clause length could be random or another fixed value. This is generalised as k-SAT and is NP-complete. The defining parameters of a k-SAT formula are the size of the set of all variables N and the total number of clauses L.
When k-SAT problems are randomly generated, the set of Booleans for each clause is picked randomly from the larger set of all available Booleans. The deciding value whether a certain problem is solvable has been found to be the ratio of clauses to Booleans $\frac{L}{N}$ [3]. The number of clauses and that of all Booleans can be seen as adversaries. On the one hand, more Booleans reduce the likelihood by which a specific Boolean occurs. With a less frequent appearance the value it can have to satisfy the formula is less restricted. Such free variables can adjust to those that are restricted and this make the problem under-constrained. On the other hand, the more clauses a formula has, the more often a specific Boolean is likely to appear. This leads to fewer free variables and makes the problem over-constrained. Since these two values work against each other, their ratio is suited to define the hardness of a problem.
It has been shown and experimentally confirmed [3] that at a certain ratio of number of clauses to number of total Booleans a phase transition occurs. For 3-SAT this value is 4.24 [4]. The example formula above has a ratio of $\frac{2}{4} = \frac{1}{2}$ which predicts it to be easily solvable. A random formula with 34 clauses and 8 different Booleans would be close to the phase transitions and therefore it would probably be hard to decide whether or not it has a solution.
In phase transitions, the problems go quickly from under-constrained (and therefore easily solvable) to over-constrained (and therefore easily determined as unsolvable). In between these two regions lies the phase transition where the problems are neither over-constrained nor under-constrained and therefore difficult to decide. The performance of solving algorithms shows an easy-hard-easy pattern where the interesting and hard instances lie in the region of the phase transition.
Models of physical problems, namely the diluted spin glass model [5], have been used to examine the phase transition in 3-SAT.

2.2.2 Related Work and Importance

Investigating phase transitions in the solvability of problems is a way to gain knowledge about that domain of problems. When the deciding parameters are known, problems can be analysed and a statement can be made whether the problem is close to the value of the phase transition and therefore possibly hard or away from this value and most likely easy. A solver can choose the used algorithm based on such information, using different approaches for very hard problems.
Rintanen [6] suggests that because search algorithms are tested on sets of standard problems, called *benchmarks*, these sets also define which algorithm is deemed good and which is not. Insights into phase transitions could therefore be adopted when creating new benchmarks to obtain sets of hard problems. Such improved benchmarks facilitate the work on better algorithms.
SAT is NP-complete and phase transitions have been found in other problems of this class, for example graph colouring [7]. Phase transitions have also been found for QSAT [8] (quantified satisfiability problems), which is PSPACE-complete. QSAT problems do not simply search for any satisfying configuration of a formula to exist but for example ask whether or not a specific variable can have any value to satisfy the formula.

2.3 Sokoban

Sokoban is a puzzle game that was developed in 1981 by the Japanese programmer Hiroyuki Imabayashi in the language BASIC and published in the following year by the company Thinking Rabbit [9]. Today, many versions of Sokoban exist with implementations for many devices.
Sokoban is Japanese for warehouse worker. This worker is the only agent in the game and controlled by the player. The warehouse is displayed as a grid of square fields and seen from an aerial perspective. The agent can move from one free field to another vertically and horizontally but not diagonally. Each level is different. The levels are surrounded by walls and they can also be within the level, usually shaping a maze. They are impassable and unmovable. Each level contains at least one, usually several boxes, also called stones, and the same number of goal fields.
The objective is to push all boxes onto goal fields. Boxes cannot be pulled and only one box at the same time can be moved, meaning that the space the box is pushed to must be free before it is pushed. Thus a field can only be occupied by one box at the same time and the player cannot be on the same field as a box. It does not matter which box ends up on which goal field, as long as all boxes are stowed away. Note that boxes can be on a goal field in the initial state. In man-made levels this most often means thats the box has to be pushed away to push other boxes through that space and later the first box has to be pushed back to the goal field. Figure 2.1 shows a man-made Sokoban problem.

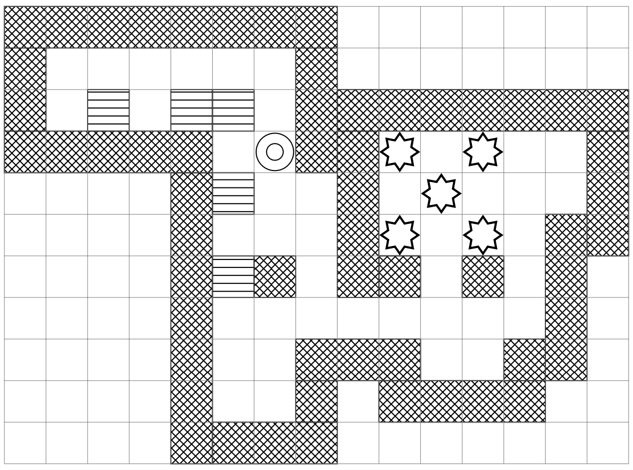

Figure 2.1: A man-made Sokoban level. The stars are the goals, the hatched squares are boxes and the circle is the agent.

To investigate phase transitions for Sokoban, problems are created at random for this thesis. While man-made problems often consist of elaborate mazes and contain goal areas where many goal fields are right next to each other, this is very unlikely to happen by chance. Figure 2.2 shows such a randomly generated Sokoban. Solving man-made levels often comes down to recognising the creators intent and finding the idea and pattern behind it.
A particular challenge of the game is that the boxes and the player can get stuck easily. When a box is pushed towards a wall there is no way to move the box back away from the wall again, unless the wall leads to an open space which possibly frees the box. There is no

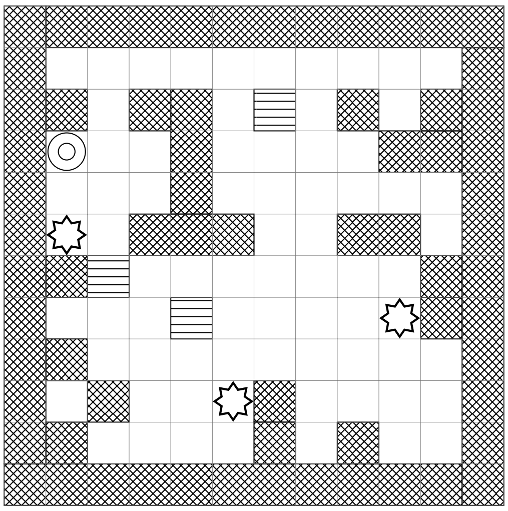

Figure 2.2: A randomly generated Sokoban level.

recovery from such situations except to restart the level. Such states are called *deadlocks* and are sometimes hard to identify. Designing challenging, yet solvable levels is the main task for creators. The game is most popular in Japan where the best level creators are seen as artists [10]. Collections of such interesting levels can be found online [11].
Sokoban specific solvers have been written, notably Junghanns' and Schaeffer's implementation [12] called *Rolling Stone*. Published in the year 2000, it uses domain specific knowledge about Sokoban to improve its search. The most recent work is that of Pereira, Ritt and Buriol [13] who utilise pattern databases recognising known abstract similarities between levels.

2.3.1 Generalisation

Dor and Zwick [14] generalised the Sokoban problem by allowing any number of boxes to be pushed and pulled simultaneously. They called this family the *Motion Planning Problems*. This category should not be confused with motion planning in general which is part of robotics. This family of problems requires one or more agents that interact with a world of objects, obstacles and goals to transform an initial situation into a desired goal state. In their notation, $Sokoban(k, l)$ stands for a Sokoban problem where up to k boxes can be pushed at the same time and l boxes can be pulled at the same time. The normal Sokoban would be called $Sokoban(1, 0)$ in this notation, since there is no pulling and only one box can be pushed at the same time.
Instead of being square, fields can have the shape of any regular polygon and the according number of fields around them (although tessellation can only be reached with triangles, squares and hexagons). For example, in Hexoban [15] the fields are hexagons and each field has six neighbours which increases possible moves. In some versions of Hexoban the boxes are time bombs and the levels have to be solved in time to make the game dramatic.

2.3.2 Computational Complexity

The reason why Sokoban is not just a puzzle game but also of interest to artificial intelligence and automated planning is its complexity. Historically, many man-made levels could not be solved by computers at all. This is true especially for bigger levels. Although this has changed with domain specific solvers and today's computing power, the problem remains demanding. Some levels that would be seen as trivial for humans are difficult for machines due to the huge number of possible move sequences.
An impression of Sokoban's complexity can be gained when reflecting on the possible states the puzzle can be in. Since walls and goals are fixed, each different position for the boxes and the player is a different state. Because it does not matter which box is on a specific place, combinations are asked, not permutations. The place for the m boxes can be chosen out of n non-wall fields. The player then has to be on one of the remaining $n - m$ fields. The number of states S can be computed by the following formula:

$$S = \binom{n}{m} * (n - m)$$

For 144 free fields and 10 boxes this gives approximately 10^{17} states. These reflections are confirmed by Junghanns' calculations [10]. The number grows fast with the number of free fields and is maximized when the number of boxes is half of the number of non-wall fields. Junghanns deduces 10^{98} states for the size of 18 and 50 percent boxes. However, a level half full with boxes is only solvable in the best man-made cases (with some boxes already on goal fields). So for solvable, randomly generated Sokobans, the number of theoretical states is about as high as the one for the famous Rubik's Cube problem which has $4.3 * 10^{19}$ [16] states.
For the solver not all of these states are relevant because not all are generally reachable from a given starting state. This number is therefore an overestimation. Finding out which states can be reached legally in a certain Sokoban is the same as looking for a plan where the goal state is the investigated state. This means creating the collection of all reachable states, called the *state space*, would mean to solve the problem for each theoretically possible state. In other problems, such as Rubik's Cube, the state space remains the same for each problem and can be calculated. Many different states can be linked to the same abstract state for symmetric reasons. Even though these states are different on paper, they can be progressed to the goal state by using the abstract state they are represented by. In Sokoban each unique problem has its own unknown state space.
The successor states of a state are called its *children*. The number of children per state is called the *branching factor*. Sokoban has a high average branching factor because in most states the player can move freely, opening new possible branches that need to be regarded. Since moving does not add to the total cost, only pushes can be seen as relevant. With 10 boxes that are free and can be pushed in any of up to 4 possible directions, the branching factor would be 40. This is a maximal estimation for an average problem, but the average value would be lower and dependent on the specific problem.
The search space increases with each step by the current branching factor. After only five moves this leads to possibly $40^5 \sim 10^8$ states but this number is greatly reduced during the search by finding and eliminating identical states that were merely reached in different ways. Nevertheless, these considerations and numbers illustrate Sokoban's difficulty. Sokoban has

been shown to be PSPACE-complete by Culberson [17].

2.3.3 Comparison to k-SAT

There are similarities between Sokoban and the Boolean Satisfiability problem. In k-SAT, the number of clauses restricts solutions and makes the problem less likely to be solvable for a fixed set of variables. The same can be said for walls in Sokoban. A level with many walls that block paths can be seen as over-constrained. The same is true for boxes, but they are not equal to walls.

On the one hand, we can expect boxes to be worse for the solvability than walls because they must all be reachable. Walls can be in regions that are irrelevant to the solution of the problem, because there are neither boxes nor goals there. Unlike the other walls, these walls have no influence on the solvability. This lets us conclude that the average wall has less influence than any box.

On the other hand, we can expect them to be not as bad for the solvability as walls, because they are movable. These thoughts are pursued further in the chapters 3.4 and 4.

2.3.4 Sokoban in PDDL

The notation utilised for this thesis is the Planning Domain Definition Language (PDDL). This language has been created to standardise the representation of planning problems. Different kinds of problems can be defined by specifying their *domain*. This makes it possible for planners that understand PDDL as an input to solve problems of many different domains. The domain, which is the same for all problems of the same kind, defines the fundamental properties for problems of this kind. It defines types which can be concrete concepts like boxes or mere categories like objects which boxes are a part of. For Sokoban these are *thing*, *object*, *location*, *direction*, *stone* and the *player*.

It further declares the predicates of the domain, which are the different states that the types can be in. It also defines which types are required for a predicate. For Sokoban these predicates are to be *clear*, to be *at* a location, to be *at a goal*, to *be a goal* or *not to be a goal* and for a specific *move direction* to be available from a particular location to an other. While "clear" only describes one location, "move direction" describes two locations and a direction at once.

The actions can be seen as functions that have input parameters, preconditions and an effect. The entirety of all actions define the rules of the domain and how the world can be manipulated. In this example the actions are: *move*, *push a box to a non-goal field* and *push a box to a goal field*. The last two actions need to be differentiated because the goal state is defined by the boxes having the predicate "at goal" and not by their location (if that was the case, all possible combinations would have to be enumerated as goal states). Pushing a box has the precondition that the player is next to the box and that the field behind it is free. Its effect is that the box's and the player's locations are changed respectively and the total cost is increased by one.

A specific problem conforms to a domain if the domain language is used to describe itself. It lists which instances of which types occur and gives them a name. For Sokoban these are the directions, the locations, the boxes and the player. It continues by defining the initial state by assigning predicates to the types. In Sokoban this means stating which locations are goals and which are free, where the player and the boxes are and which move directions

are available from each location. It concludes by defining the goal states and which function should be minimised or maximised for optimisation. For Sokoban there is always only one goal state: every box needs the predicate “at goal”.
The following example is the PDDL notation of the smallest Sokoban problem that is not already solved. Figure 2.3 visualises this level. Note that the box is called “stone” in this domain. This file is accompanied by the domain file which can be found in the Appendix.

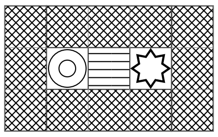

Figure 2.3: The smallest Sokoban level.

```
(define (problem example-sokoban)
  (:domain sokoban-sequential)
  (:objects
    dir-down - direction
    dir-left - direction
    dir-right - direction
    dir-up - direction
    player-01 - player
    pos-01-01 - location
    pos-02-01 - location
    pos-03-01 - location
    stone-01 - stone
  )
  (:init
    (IS-GOAL pos-03-01)
    (IS-NONGOAL pos-01-01)
    (IS-NONGOAL pos-02-01)
    (MOVE-DIR pos-01-01 pos-02-01 dir-right)
    (MOVE-DIR pos-02-01 pos-01-01 dir-left)
    (MOVE-DIR pos-02-01 pos-03-01 dir-right)
    (MOVE-DIR pos-03-01 pos-02-01 dir-left)
    (at player-01 pos-01-01)
    (at stone-01 pos-02-01)
    (clear pos-03-01)
    (= (total-cost) 0)
  )
  (:goal (
    (at-goal stone-01)
  ))
  (:metric minimize (total-cost))
)
```

Figure 2.4: PDDL of the smallest Sokoban level.

3 Investigating Phase Transitions

Phase transitions have been found for other PSPACE-complete problems [8], which leads us to the assumption for this work that they can also be found for Sokoban. This chapter describes how phase transitions are investigated. To get statistical data on the solvability random problems are generated. Section 3.1 presents the random Sokoban generator in detail. Section 3.2 elaborates on the decisions for the experiments in which big numbers of problems are processed. Section 3.3 states the tools and methods for analysing the experiments. Section 3.4 describes the attempt to unify the parameters and the idea behind it.

3.1 Generating Random Problems

A generator for random Sokoban problems has been written in Java. The deciding inputs given to the generator are: *size*, *ratio of walls* and *ratio of boxes*. The size can be set by length and width but for this thesis only square Sokoban are investigated further. The ratios are seen as how many of the total fields are boxes or walls. The Sokoban is represented by a matrix of fields, each possessing Boolean values for being a wall or a box or the player and/or a goal field, while free fields hold all these values false.
After checking the input for legality (at least one box, enough space for walls and boxes, legal size etc.), the corresponding number of walls, boxes, the same number of goals and one player is randomly placed within the Sokoban. The output is then transformed into PDDL and written into a file. The file starts with a visualisation of the level [18] using ASCII characters as graphic representation. While the planner only cares for the actual PDDL part, humans can use it to understand the properties of the level in one glance and make spot tests for found plans.

3.1.1 Eliminating Trivially Unsolvable Problems

An additional property for the Sokoban generator makes it possible to prevent trivially unsolvable problems. Even wall-less instances can be unsolvable because boxes are placed on the edge of the field. These boxes cannot be moved and thus the problem is only solvable if there happens to be a goal placed on the same edge.

To normalise the solvability, all the easiest problems are made solvable by preventing boxes from being placed on the initial edges. These are called *fair* Sokoban problems. The solvability for fair Sokobans ranges from one to zero with the easiest problems being always solvable and after a certain point of difficulty none of the problems being solvable.
Discarding such trivially unsolvable instances is in accordance to previous work on phase transitions for PSPACE-complete problems. Gent and Walsh [8] conclude in their work on QSAT (quantified satisfiability) that such *flawed* problems, as they call them, should be disallowed.
Note that fair Sokobans can still be unsolvable due to boxes that are unmovable because of inner walls but for a wall ratio of zero, fair problems will be solvable, unless they contain so many boxes that it is possible for them to be completely blocked by each other. Many boxes that are next to inner walls can be moved to few fields. Checking each of them on its ability to reach a goal field would already mean in parts to solve the problem and could not be treated independently from all other boxes. These unsolvable instances are therefore not deemed trivial.

3.2 Designing Experiments

To reduce random noise, large numbers of random problems with the same properties are created. An *experiment* is a set of problems for which two of the three parameters (size, wall ratio, box ratio) are fixed and one is iterated through sensible values. For each iteration point a number of problems is generated with the same parameters. The standard error on early experiments shows that a thousand instances for each set of parameters give reliable data. Experiments that all iterate the same first parameter through the same values are collectively a *series of experiments*. Within the series a second parameter is iterated while the last one remains fixed for the entire series. For example, a series of experiments has the constant problem size of 11 and goes from a box ratio of 0.2 to 0.9 in steps of 0.1. Within each experiment the ratio of walls runs from 0 to 0.48 in steps of 0.04. Such and other series give an insight into the effect of boxes and walls on the solvability of problems. Other experiments and series of experiments kept different values fixed while iterating others.
The set of all possible combinations of parameters for Sokoban can be seen as the problem space. It is stretched by the axes size, ratio of walls and ratio of boxes. Keeping one value fixed leaves a plane in this space that can be investigated in interesting regions. Series of experiments are lines on such planes.
The problems are solved by the planning system Fast Downward [19]. Since the thesis is focussed on solvability and not optimisation, the Fast Downward's greedy search algorithm is used with the h^{FF} - Heuristic. For scripting the experiments the python library Lab [20] is utilised. To maintain a stable and invariant test environment the experiments are conducted on the Maia grid of the Universitätsrechenzentrum Basel.

3.3 Recognizing Phase Transitions

Lab captures the outcome of each experiment into a file. The data from these JSON files is extracted, analysed and visualised using Python scripts with the libraries NumPy, SciPy and Matplotlib. The investigated qualities for phase transitions are: *coverage of finished* and the *average search time*.

While solvability is the theoretical value for the ratio of solvable problems to all problems, coverage is the measured value for actually solved problems. This is not necessarily identical since not all theoretically solvable problems might actually be solved.
When analysing the coverage of any number of problems with the same parameters there is a difference between the coverage of total runs and the coverage of finished runs. While the coverage of total gives the ratio of solved problems relative to all problems given to the planner, the coverage of finished only considers runs whose searches came to an end. The coverage of total therefore counts unfinished runs as unsolvable where the coverage of finished simply ignores them. Both decisions can slightly distort the outcome.
On the one hand, the coverage relative to all searches counts problems whose searches did not finish in time as unsolvable. But some of these problems might have been solvable given enough time. On the other hand, the coverage relative to finished searches discards unfinished problems. However, for the discarded, unfinished problems the probability to be solvable is below the average probability of the rest, because the longer a search is, the less likely it will come to a positive end. This is confirmed in this work by comparing solvability graphs with increased maximal search time limitations. Therefore, the coverage of all runs slightly underestimates the actual solvability and the coverage of finished runs slightly overestimates it.
In the best case there are no unfinished runs and while the goal of extending the maximal search time to 30 minutes per problem is to achieve just that, it cannot be guaranteed in all cases. However, only finished searches are considered for the detection of possible phase transitions.
The standard errors in coverage and search time are calculated with the statistical package in NumPy. This gives an insight into how volatile the results are due to the random creation of the problems.
The data is illustrated by creating graphs that show the coverage on one axis and the iterated parameter on the other. To find exact values that lie between the experimentally found values, linear interpolation is applied. Phase transitions from solvable to unsolvable occur in regions, rather than exact points, yet the point of fifty percent coverage is found to be the most stable choice in defining a phase transition. Hence, it will also be referred to as the *transition point*. Other ideas like considering the steepest or the inflection point are rejected because they lack stability due to the statistical noise.

3.4 Unifying Parameters

When investigating phase transitions for SAT, Gent and Walsh [3] put the defining properties of such a problem, the number of clauses and the number of literals, into relation to one another (see chapter 2.2.1). The major insight is that regardless of the size of the problem or the number of variables, the probability to create a solvable instance only depends on the ratio between them. The phase transition and therefore the hard problems occur at a specific value for this ratio. Problems with a higher ratio are predominantly easy and unsolvable and those with a lower ratio predominantly easy and solvable. To find such a defining value for randomly generated Sokoban, a unifying formula for all parameters is constructed.

4 Results

This chapter presents the results that are relevant for the conclusions. Section 4.1 shows experiments in which the ratio of walls is the iterated parameter. The visualisation of an entire series of experiments with varied box ratios follows. Section 4.2 presents the results of measuring the difference between the influence of boxes and that of walls and present the formula for the combined ratio. Section 4.3 then shows how this formula is used. Section 4.4 presents an experiment in which the size was iterated.

4.1 Varying the Ratios of Walls

Figure 4.1 shows an experiment with Sokoban problems of size 10 with 4 boxes (which in this case is the same as 4 percent of boxes). The percentage of walls is raised in steps of 4 from 0 to 48 and for each value 1000 instances are given to the planner. The coverage graph

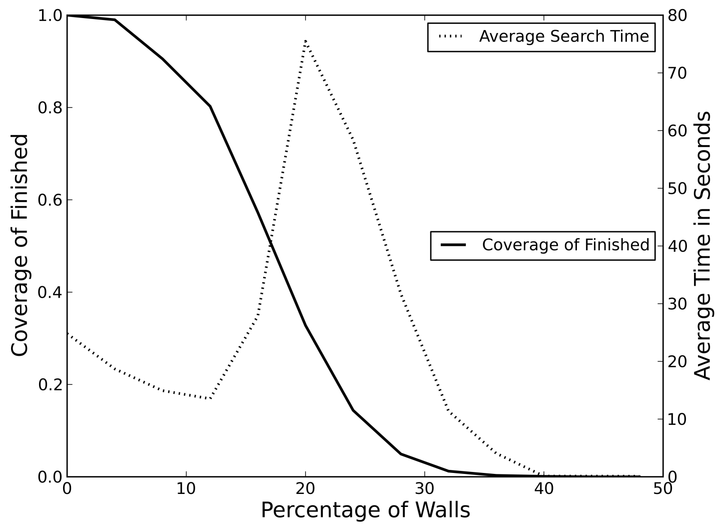

Figure 4.1: An experiment with an iterated wall ratio. The solid line represents the coverage of finished searches, the dotted line represents the average search time.

represents the ratio of problems for which the planner has found a solution relative to all problems with finished searches. The search time graph represents the arithmetic mean of the time the planner needed to finish these searches. The scale on the left is for the coverage which is between 0 and 1. The scale on the right is for the average search time in seconds.
The coverage graph starts at 1 for no walls and decreases with increasing ratios of walls. The graph shows only a small decline in the beginning but then drops fast. Its steepest decrease is in the region of about 12 to 20 percent walls. 50 percent coverage is reached at a wall ratio of 0.172. After this region the coverage soon goes below 15 percent where the graph flattens. It approaches 0 at about 35 percent walls. For higher ratios of walls the rare, solvable instances that occur have most boxes already on a goal field at the beginning.
At the same time, the average search time peaks at 20 percent walls. There are two important things to note about the search time. Firstly, the highest average search time occurs at the end of the region with the steepest decline in coverage. Problems after that region (for example for 24 percent walls) have a higher average search time than those right within that region. The second interesting thing is that the average search time is high for a wall ratio of 0 and first declines up to the point of 12 percent walls where the steep decline in coverage begins.

4.1.1 Increasing the Boxes

Figure 4.2 shows a plot of a series of experiments for which the number of boxes is raised from experiment to experiment. They all have a problem size of 10 but differ in the number of boxes to each other.

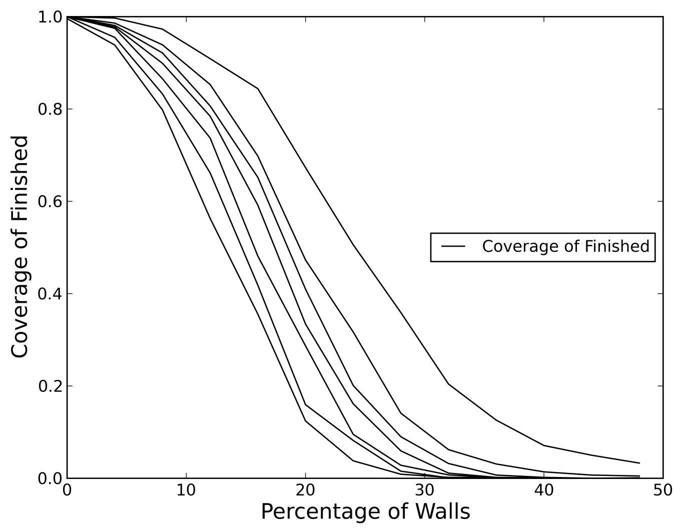

Figure 4.2: A series of experiments with iterated boxes. Each of the lines represents one experiment. The one on the right has 1 box. The number of boxes increases for each graph going left. The one on the left represents an experiment with 7 boxes.

What can be seen is that the coverage curve is shifted for each additional box. For more boxes the region with the steepest decline occurs at lower wall ratios.
The point of 50 percent coverage occurs at 24.2 percent walls for 1 box. For 2 boxes the point jumps to 19.5 percent walls and from there on the decline per box is approximately

constant. The exact value of the average decline of wall ratio per additional box ratio is important and calculated later.
There are two things to take from this that are very important. The first insight is that increasing the boxes causes an approximately even shift of the steep region and the point of 50 percent coverage to lower ratios of walls. The second insight is that the graph for 1 box gets out of line with the others. The difference from 1 to 2 boxes is unlike any other step and does not show a uniform pattern which applies for the rest.

4.2 Difference in Influences

Since both the ratio of boxes and the ratio of walls are constraints the basic idea is to sum them up to a *combined ratio*. Each ratio is differently influential on the solvability, as was to be expected. But surprisingly a linear correlation of their influences can be detected empirically within certain boundaries. Figure 4.3 shows the points with 50 percent coverage for a series of experiments. The values are taken from the first discussed series which had a fixed size of 10, an iterated wall ratio within the experiments and an iterated box ratio within the series.

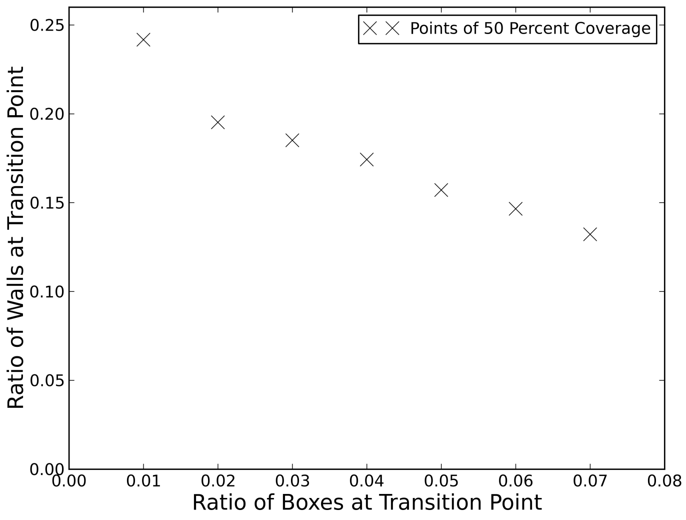

Figure 4.3: Points with 50 percent coverage for a series of experiments. Each cross marks one experiment. The x-axis shows the box ratio of the experiments and the y-axis shows the measured wall ratios at their points of 50 percent coverage.

The value for 1 box gets out of line with rest and therefore we ignore it. The rest of the points show a linear connection. They are seen as samples of a linear function and a line is interpolated through them. Figure 4.4 shows the linear interpolation of the previous values. The gradient of this graph is the negative value of the *constant* which is discussed below. This value describes how much more negative influence boxes have on the coverage than walls. Its absolute value is greater than 1 which means that boxes are more damaging for the coverage than walls for random Sokoban problems of size 10.

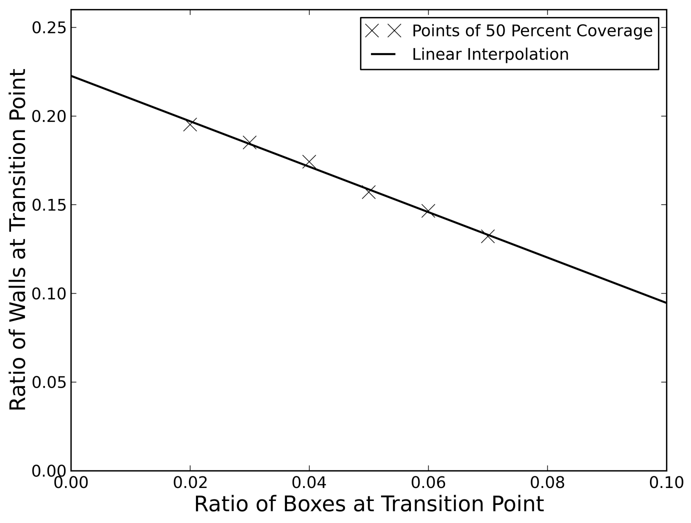

Figure 4.4: Linear interpolation is conducted for the points with a box ratio form 0.02 to 0.07.

4.2.1 The Unifying Formula

The unifying formula is designed as a linear combination of the ratios. This means that one of the two ratios is multiplied with a constant and then added to the other resulting in the *norm*. The box ratio was chosen to be multiplied by the constant, because it was the one that had to lie within given limits and therefore was easier to be chosen first.
The constant describes the difference in the influence of the wall ratio versus the box ratio. The constant depends on the size of the problem and while not all sizes were investigated, we can assume that there is a function that links each size to its designated constant. This hypothetical function f has the size s as input and provides the corresponding constant c. It is multiplied with the ratio of boxes b and added to the ratio of walls w resulting in the norm N. This norm for randomly generated Sokoban problems makes a direct prediction on the instance's solvability.

$$N = f(s) * b + w$$

The generator has been updated to allow the creation of problems with a *combined ratio*. This is the norm for an empirically determined constant. The values for the two ratios are chosen randomly but in a way that they add up to the desired combined ratio. The restriction is that the ratio of boxes must result in at least two and at most seven boxes. This is because linearity did not occur for one box and was not empirically verified for more than seven boxes. With this combined ratio we can make experiments that have neither a fixed ratio of walls nor of boxes.

4.2.2 Constant and Norm Values

Three different series of experiments have been conducted for problems of the size of 10. The constant is calculated as the (negated) slope of the interpolation previously shown. The found values are: 1.280, 1.271 and 1.294. This shows that the empirical value is not exact

but we can say that the actual value is close to these values. For the size of 9 the value from one series is 1.064.
When we substitute the points of 50 percent coverage back into the formula we get the norm value for which we can expect 50 percent coverage which is 0.22. This value is the same as the y-intercept of the interpolated line. We can expect a higher coverage if the problem's norm is below this value and a lower coverage for problems that have a norm above this value. The prediction on the hardness of a random Sokoban is that the problem is hard if its norm is close to 0.22 and easy if its norm is away from this value.

4.3 Applying the Combined Ratio

Figure 4.5 shows the coverage of an experiment with an iterated combined ratio for problems of the size 10. The used constant was 1.27.

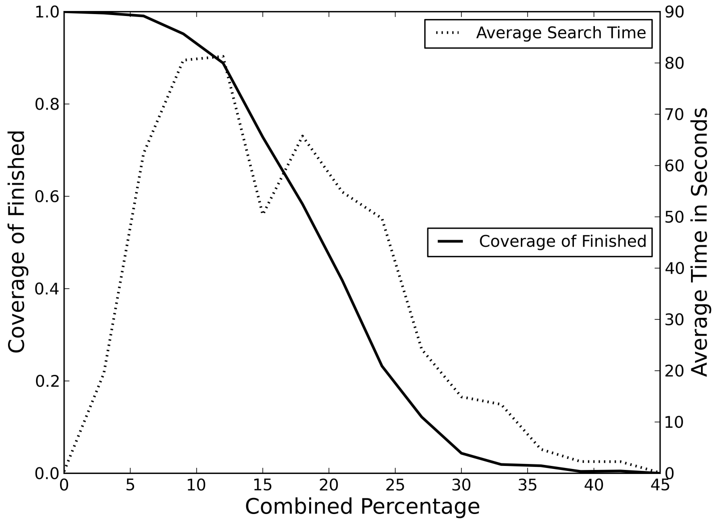

Figure 4.5: An experiment with an iterated combined ratio and a size of 10. The solid line represents the coverage of finished searches and the dotted line the average search time.

The coverage shows a region of steep decline between 15 and 25 combined percent. The average search time graph has peaks at 12 and 18 combined percent but also has a local minimum at 15 combined percent. Note that while the standard error for all shown coverage values is very low, it is very high for these values for the average search time.
The predicted value for the point of 50 percent coverage is not quite confirmed empirically. We expect this point to be at a combined ratio of 0.22 but the measured value is 0.195. However the graph shows great similarities to graphs with fixed boxes. This shows that the formula works in combining the ratios and reducing the parameters. If the constant was known for all sizes we could create problems for which the size is also random. The constant would then be chosen according to the size and the only input would be the norm.

4.4 Varying the Size

The idea for this is to set the value for the constant to 1.27 and use a fixed combined ratio of 0.22. These values have been found for Sokobans of size 10 and we now apply them blindly for Sokoban problems of other sizes. If the result was that coverage is always approximately 50 percent then this would mean the value of 1.27 is universal and independent of the problem size. We show that this is not the case.
Figure 4.6 shows an experiment with a fixed combined ratio of 0.22 and an iterated size. The size goes from 5 (meaning 5 x 5 Sokoban problems) to 32 in steps of 3.

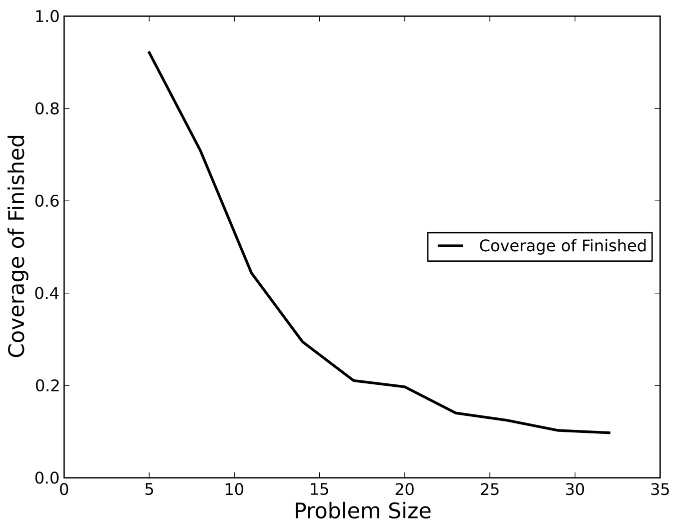

Figure 4.6: An experiment with a combined ratio of 0.22 and an iterated size. The solid line represents the coverage of finished searches.

Applying the constant for size 10 on smaller problems leads to more than 50 percent covered problems. For sizes close to 10 it leads to about 50 percent coverage. If random Sokoban problems that are bigger than size 10 are created with the constant found for size 10, then less than 50 percent coverage results. The point of 50 percent coverage lies at an interpolated value of 10.4 which further confirms the prediction of the norm.
This means that for smaller Sokoban problems the constant must actually be lower. Through the combined ratio the constant of 1.27 leads to more than 50 percent solvable problems. Therefore the influence of boxes is overestimated for smaller sizes and the actual value for the constant would be below 1.27. It might even be below 1 since the found value for size 9 was already 1.06. A value lower than 1 would mean boxes are less damaging for the solvability than walls.
The opposite is true for problems with a bigger size than 10. The constant of 1.27 leads to less than 50 percent covered problems. This means that the influence of boxes on the solvability is underestimated and the actual value for the constant would be above 1.27.
The graph of this experiment gives an idea of how the constant adjusts to an iterated size. It is a conjecture of how the described function f that transforms a size into its respective constant could look like.

5 Conclusions

Figure 4.1 shows a phase transition from solvable to unsolvable instances at a wall ratio of about 0.172 for Sokoban problems with a length and width of 10 and 4 boxes. The ratio of walls per fields was iterated and the phase transition occurs in the region from about 12 to 20 percent walls. The average search time has its maximum at 20 percent walls which suggests that the hardest problems lie there.

An open question is how steep and narrow a region of fast decline has to be to qualify as a phase transition. This graph is not as steep as the one for SAT [3]. However this might be because Sokoban is PSPACE-complete. The found phase transition for QSAT [8], which is PSPACE-complete itself, is also not as narrow and steep as the one for SAT. The correlation with the peak in average search times lets us conclude that the shown behaviour actually qualifies as a phase transition.

Figure 4.2 shows that, aside from the first box, adding boxes causes an even shift of the phase transition to lower wall ratios. Therefore boxes have a similar influence as walls and the connection between them is approximately linear.

Figure 4.4 shows the transition points for a series of experiments with an iterated ratio of boxes and a size of 10. The wall ratio of the points declines evenly with increasing box ratio. Linear interpolation gives a gradient of -1.28 and a y-intercept of 0.22.

The gradient expresses the relation between the influences of boxes and walls. We can say that adding a box is about 28 percent more damaging to the solvability than adding a wall for random Sokoban problems of size 10. Its negated value is the constant by which the box ratio has to be multiplied to adjust it to the wall ratio.

Since the two constraining ratios have a linear connection, the formula for the Sokoban norm is a linear combination as described in Section 4.2.1. This norm unifies the parameters for a Sokoban problem (size, wall ratio and box ratio) to one value. This value defines the solvability of randomly generated problems.

When substituting the transition points back into this formula, we get the transition norm, which is 0.22. Problems with a norm lower than 0.22 are predominantly solvable and problems with a higher norm predominantly unsolvable.

Figure 4.5 shows a phase transition for an experiment with an iterated combined ratio. The values for the box and wall ratio are random but add up to the desired norm. This is done

with the above mentioned formula and empirically measured values for the constant. The phase transition occurs in a region close to a combined ratio of 0.195. The formula for the norm can therefore be applied.
Figure 4.6 shows an experiment with a fixed combined ratio of 0.22 and an iterated size. The used constant was 1.27. These values give 50 percent solvable problems for the size 10. Blindly applying it to different sizes has the following effect: For problems smaller than 10 a coverage greater than 50 percent was measured and for problems bigger than 10 a coverage of below 50 percent was measured. This lets us conclude that the constant is indeed dependent on the size. We assume that it is below 1.27 for smaller problems and above 1.27 for bigger problems.
An important thing to note about this work is that there is no standard way to investigate phase transitions that can be applied blindly to every domain. Each domain must be approached differently and while ideas and methods from other phase transition investigations can be used, they need to be adapted to the domain.

5.1 Future Work

The most interesting question to answer would be whether the found linear connection between the influences of boxes and walls really occurs for all sizes. The searched region of the problem space consists of problems that are 9 by 9, 10 by 10 or 11 by 11 and a ratio of boxes from 0.02 to 0.09. Can we confirm that the assumed linearity between walls and boxes also occurs outside of this region? More series of experiments would be required to confirm and test or possibly falsify the found values and concepts.
Another possibility would be to discard the idea of using the ratio of boxes and walls completely. This concept does not directly reflect that each level is surrounded by walls. This means small levels are more restricted. Not only because they have fewer fields overall but because they have more walls per field. The number of edge fields grows linear $(4 * n - 4)$ but the number of inner fields grows quadratic $(n - 2)^2$. This means that the bigger the level the fewer implicit walls per field there are.
Problems could also be defined by their number of total free fields and their boxes. However this idea has the disadvantage that there is no measurement of how narrow the spaces are within the Sokoban. A level with 100 free fields and no walls is solvable while a level with 100 free fields and 100 walls most likely is not. The narrowness could be measured by the average number of neighbouring fields that are a wall. This would then lead to three alternative defining parameters: the number of free fields, the number of boxes and the average number of neighbouring walls per field. This would account for the surrounding walls each level has.
Investigating measurements for the use of resources other than the search time would also be interesting. We could use the found norm to create sets of problems that are predicted to be hard. Those sets that are cleaned from unsolvable instances could be used as benchmarks for planners.
Finally, a possible future work would be to investigate phase transitions for other PSPACE-complete problems aside from Sokoban and draw a comparison to this work.

Bibliography

[1] Hogg, T., Huberman, B. A., and Williams, C. P. Phase Transitions and the Search Problem. *Artificial Intelligence*, 81(1):1–15 (1996).

[2] Arratia, R. and Waterman, M. S. A Phase Transition for the Score in Matching Random Sequences Allowing Deletions. *The Annals of Applied Probability*, 4(1):200–225 (1994).

[3] Gent, I. P. and Walsh, T. The SAT Phase Transition. In *11th European Conference on Artificial Intelligence*, pages 105–109 (1994).

[4] Crawford, J. M. and Auton, L. D. Experimental Results on the Crossover Point in Random 3-SAT. *Artificial Intelligence*, 81(1):31–57 (1996).

[5] Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. Determining Computational Complexity from Characteristic Phase Transitions. *Nature*, 400(6740):133–137 (1999).

[6] Rintanen, J. Phase Transitions in Classical Planning: An Experimental Study. In *International Conference on Automated Planning and Scheduling*, pages 101–110 (2004).

[7] Hogg, T. and Williams, C. P. The Hardest Constraint Problems: A Double Phase Transition. *Artificial Intelligence*, 69(1):359–377 (1994).

[8] Gent, I. P. and Walsh, T. Beyond NP: The QSAT Phase Transition. In *Association for the Advancement of Artificial Intelligence*, pages 648–653 (1999).

[9] Imabayashi, H. Sokoban (2013). URL `http://www.sokoban.jp`. Retrieved: 16-12-2013.

[10] Junghanns, A. *Pushing the Limits: New Developments in Single-Agent Search*. Ph.D. thesis, University of Alberta (1999).

[11] Myers, A. XSokoban (2001). URL `http://www.cs.cornell.edu/andru/xsokoban.html`. Retrieved: 16-12-2013.

[12] Junghanns, A. and Schaeffer, J. Sokoban: Enhancing General Single-Agent Search Methods Using Domain Knowledge. *Artificial Intelligence*, 129(1):219–251 (2001).

[13] Pereira, A. G., Ritt, M. R. P., and Buriol, L. S. Finding Optimal Solutions to Sokoban Using Instance Dependent Pattern Databases. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search*, pages 141–148 (2013).

[14] Dor, D. and Zwick, U. SOKOBAN and Other Motion Planning Problem. *Computational Geometry*, 13(4):215–228 (1999).

[15] Sunshine, E. SokoSave (2011). URL `http://www.high-speed-software.com/sokosave/links`. Retrieved: 16-12-2013.

[16] Korf, R. E. Finding Optimal Solutions to Rubik's Cube Using Pattern Databases. In *Association for the Advancement of Artificial Intelligence*, pages 700–705 (1997).

[17] Culberson, J. Sokoban is PSPACE-complete. In *Proceedings in Informatics*, volume 4, pages 65–76 (1999).

[18] Minklei, F. Sokoban Wiki (2010). URL `http://www.sokobano.de/wiki`. Retrieved: 16-12-2013.

[19] Helmert, M. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246 (2006).

[20] Seipp, J. Lab (2013). URL `http://lab.readthedocs.org`. Retrieved: 16-12-2013.

[21] Junghanns, A. University of Alberta Sokoban (1999). URL `http://webdocs.cs.ualberta.ca/~games/Sokoban`. Retrieved: 16-12-2013.

[22] Botea, A., Müller, M., and Schaeffer, J. Using Abstraction for Planning in Sokoban. In *Computers and Games*, pages 360–375 (2003).

Appendix

The Sokoban domain in PDDL.

```
(define (domain sokoban-sequential)
  (:requirements :typing :action-costs)
  (:types thing location direction - object
          player stone - thing)
  (:predicates (clear ?l - location)
       (at ?t - thing ?l - location)
       (at-goal ?s - stone)
       (IS-GOAL ?l - location)
       (IS-NONGOAL ?l - location)
       (MOVE-DIR ?from ?to - location ?dir - direction))
  (:functions (total-cost) - number)

  (:action move
   :parameters (?p - player ?from ?to - location ?dir - direction)
   :precondition (and (at ?p ?from)
                      (clear ?to)
                      (MOVE-DIR ?from ?to ?dir)
                      )
   :effect       (and (not (at ?p ?from))
                      (not (clear ?to))
                      (at ?p ?to)
                      (clear ?from)
                      )
   )

  (:action push-to-nongoal
   :parameters (?p - player ?s - stone
                ?ppos ?from ?to - location
                ?dir - direction)
   :precondition (and (at ?p ?ppos)
                      (at ?s ?from)
                      (clear ?to)
                      (MOVE-DIR ?ppos ?from ?dir)
                      (MOVE-DIR ?from ?to ?dir)
                      (IS-NONGOAL ?to)
                      )
   :effect       (and (not (at ?p ?ppos))
                      (not (at ?s ?from))
                      (not (clear ?to))
                      (at ?p ?from)
                      (at ?s ?to)
                      (clear ?ppos)
                      (not (at-goal ?s))
                      (increase (total-cost) 1)
                      )
   )

  (:action push-to-goal
   :parameters (?p - player ?s - stone
                ?ppos ?from ?to - location
                ?dir - direction)
   :precondition (and (at ?p ?ppos)
                      (at ?s ?from)
                      (clear ?to)
                      (MOVE-DIR ?ppos ?from ?dir)
                      (MOVE-DIR ?from ?to ?dir)
                      (IS-GOAL ?to)
                      )
   :effect       (and (not (at ?p ?ppos))
                      (not (at ?s ?from))
                      (not (clear ?to))
                      (at ?p ?from)
                      (at ?s ?to)
                      (clear ?ppos)
                      (at-goal ?s)
                      (increase (total-cost) 1)
                      )
   )
)
```

UNIVERSITÄT BASEL

PHILOSOPHISCH-NATURWISSENSCHAFTLICHE FAKULTÄT

Declaration on Scientific Integrity
(including a Declaration on Plagiarism and Fraud)

Bachelor's / ~~Master's~~ Thesis *(Please cross out what does not apply)*

Title of Thesis *(Please print in capital letters)*:

____

____

____

First Name, Surname *(Please print in capital letters):* ________________________________

Matriculation No.: ________________________________

I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged.

I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

In addition to this declaration, I am submitting a separate agreement regarding the publication of or public access to this work.

☐ Yes ☐ No

Place, Date: ____________________________

Signature: ____________________________

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis .

UNI
BASEL

UNIVERSITÄT BASEL

PHILOSOPHISCH-NATURWISSENSCHAFTLICHE FAKULTÄT

Declaration on Scientific Integrity
(including a Declaration on Plagiarism and Fraud)

Bachelor's / ~~Master's~~ Thesis *(Please cross out what does not apply)*

Title of Thesis *(Please print in capital letters)*:
PHASE TRANSITIONS IN THE SOLVABILITY OF SOKOBAN

First Name, Surname *(Please print in capital letters)*: BEAT, HÄNGER

Matriculation No.: 03-066-701

I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged.

I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

In addition to this declaration, I am submitting a separate agreement regarding the publication of or public access to this work.

[✓ Yes] No

Place, Date: Basel, 18 December 2013

Signature: B. Hänger

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis .

UNI
BASEL