Artificial Intelligence 227 (2015) 52–70

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Optimal Sokoban solving using pattern databases with specific domain knowledge

André G. Pereira, Marcus Ritt *, Luciana S. Buriol

Institute of Informatics, Federal University of Rio Grande do Sul, Brazil

ARTICLE INFO

Article history:
Received 8 August 2014
Received in revised form 7 April 2015
Accepted 29 May 2015
Available online 11 June 2015

Keywords:
Single-agent search
Heuristic search
Sokoban
Pattern database
A*
Domain-dependent knowledge

ABSTRACT

A *pattern database* (PDB) stores shortest distances from abstract states to a set of abstract goal states. For many search problems the best heuristic function is obtained using PDBs. We aim to find optimal solutions for Sokoban using PDBs. Due to the domain-specific characteristics of the goal states a straightforward application of PDBs in Sokoban results in an ineffective heuristic function. We propose an alternative approach, by introducing the idea of an instance decomposition to obtain an explicit intermediate goal state which allows an effective application of PDBs. We also propose a domain-specific tie breaking rule. When applied to the standard set of instances this approach improves heuristic values on initial states, detects considerable more deadlocks in random states, and doubles the number of optimally solved instances compared to previous methods.

1. Introduction

A weighted state space problem $P = (S, A, s, T, w)$ consists of a set of states S, a finite set of actions A, an initial state $s \in S$, a set of goal states $T \subseteq S$, and a cost function $w : A \to \mathbb{R}$. Each action $a \in A$ is a function $a : S \to S$ that transforms a state into a successor state incurring cost $w(a)$. The set of goal states may be given explicitly, or implicitly by a goal condition. To a state space problem P corresponds a state space graph $G = (V, E, s, T)$ with vertices $V = S$, initial vertex s, goal vertices T and edges $E \subseteq V \times V$. An edge $(u, v) \in E$ iff $v = a(u)$ for some action $a \in A$. A state $u \in S$ has successors $\mathrm{Succ}(u) = \{v \mid (u, v) \in E\}$. The objective is to find a shortest (least cost) path in G from s to some goal state. This can be accomplished by single-agent heuristic search algorithms such as A* [1] and Iterative Deepening A* (IDA*) [2]. They explore states guided by the function $f(u) = g(u) + h(u)$, where $g(u)$ is the distance from the initial state s to state $u \in S$, and $h(u)$ is an estimate of the distance from u to a goal state. If the heuristic h is *admissible*, i.e., h never exceeds the shortest distance to the closest goal state, then these algorithms are guaranteed to find optimal solutions.

An abstraction transformation maps states u in the set of states S to *abstract states* u' in the set of *abstract states* S'. The set of actions A induces corresponding actions A', and the set goal states T a set of corresponding *abstract goal states* T'. In general, the abstraction transformation maps the set of states S into a smaller set of abstract states S', such that the distance from u' to the closest abstract goal state is at most the distance from u to the closest goal state. A *pattern database* (PDB), introduced by Culberson and Schaeffer [3], is a lookup table that stores for each abstract state u' the shortest distance to the closest abstract goal state. PDBs are typically built in a preprocessing phase by a backwards search from the set of

* Corresponding author.
E-mail addresses: agpereira@inf.ufrgs.br (A.G. Pereira), marcus.ritt@inf.ufrgs.br (M. Ritt), buriol@inf.ufrgs.br (L.S. Buriol).

http://dx.doi.org/10.1016/j.artint.2015.05.011
0004-3702/

Table 1
Search space properties of the standard set of instances of Sokoban.

	Branching factor	Solution length	Search space size	Stones	Free squares	Non-dead squares
Min.	0	97	10^8	6	49	41
Avg.	12	260	10^{18}	16	113	77
Max.	136	674	10^{31}	34	181	133

abstract goal states. The preprocessing can be time consuming, but this cost is usually amortized over the solution of several initial states of the same problem.

PDBs are an active research topic and the current best optimal solvers for several state space problems use them. Examples are the Sliding-Tile puzzle [4,5], Rubik's Cube [6], and Towers of Hanoi [7]. In this paper we study the application of PDBs to obtain optimal solutions for Sokoban.

1.1. Sokoban

Sokoban is a state space problem on a maze grid, which is defined by squares occupied by immovable blocks (*walls*) and free squares. There are k movable blocks called *stones* and k goal squares. The *man* (Sokoban) is a movable block that can traverse free squares and push stones to adjacent free squares. A solution of Sokoban is a sequence of such actions that moves the stones from their initial positions to the goal squares. The most common objective is to find a solution which minimizes the number of pushes, without accounting the moves of the man. In this paper we are interested in an admissible solver for this objective. An admissible solver always finds an optimal solution to an instance if one exists, while a non-admissible solver can return any feasible solution.

In Sokoban a state is defined by the positions of the k stones and by the *reachable component* of the man, i.e., the set of free squares reachable by the man without pushing stones. A reachable component can be represented by a normalized position of the man, e.g. the leftmost upper free square of the component. A *goal state* is defined implicitly as a state in which each stone is on a different goal square. Sokoban has $k!$ goal states, since the stones are unlabeled and can be placed on any goal square. A *deadlock* is a state $u \in S$ which is reachable from s but cannot reach any goal state. In general deadlocks are hard to detect. A particular situation of deadlocks in Sokoban is caused by *dead squares*. A free square is *dead* if a stone on it cannot be pushed to any goal square. Note that according to this definition a goal square is never dead.

Robot motion planning is a fundamental problem in robotics with a large range of applications. Sokoban is a simplified model of motion planning that computes a collision-free path between origin and destination points. There is a standard set of 90 problem instances used in the literature, ordered roughly from easiest to hardest in difficulty for a human to solve. Table 1 shows some search space properties of these instances. Sokoban is a challenging problem, and one of the remaining puzzles which humans solve better than computers. All instances of the standard set have been solved by humans, but even the best non-admissible solvers are not able to solve all of them. Sokoban is PSPACE-complete [8], and is harder to solve than other well-known single-agent search problems like Rubik's cube or the 24-puzzle, due to its large branching factor, greater solution length, larger search space size, and a more complex computation of the heuristic value [9]. Real world problem characteristics like the presence of deadlocks, states that are more complex to represent and generate, and a lack of symmetry also contribute to the difficulty of solving Sokoban. For this reason, only a few instances have been solved with admissible techniques, and most of the Sokoban solvers are concerned only with finding a solution using techniques without optimality guarantees.

1.1.1. Rolling stone

Rolling stone is one of the best known solvers for Sokoban. It comes in an admissible and a non-admissible version. Both use an IDA* search and multiple domain-independent and domain-dependent enhancements. The admissible version, which we call RS*, uses techniques like an enhanced heuristic, move ordering, tunnel macros, transposition and deadlock tables. When limited to explore 20 million nodes and with a deadlock table of approximately 22 million entries it solves six instances. RS* is currently the best admissible Sokoban solver [9]. In an attempt to solve more instances, the non-admissible version, referred to as RS, applies techniques that do not guarantee optimality such as goal cuts, pattern searches, relevance cuts, overestimation, and rapid random restart. RS is able to solve 57 instances within the same limits.

1.1.2. Other solvers

Botea et al. [10] proposed planning on an abstraction of the search space. Applied to Sokoban they obtain a non-admissible solver which is able to solve ten instances from the standard set in less than three minutes and exploring less than one million nodes. Demaret et al. [11] describe a non-admissible solver that uses a hierarchical planning strategy along with deadlocks learning to solve Sokoban. Using this approach, they were able to solve 54 instances from the standard set. In this case, the stopping criterion was eight hours of running time per instance.

The state-of-the-art non-admissible solvers have been developed by the Sokoban community. Among the ones with the best results, JSoko is able to solve 71, YASC 79, and Takaken 86 instances [12–15]. Since these results have not been published formally, it is not always clear which techniques have been used to achieve them.