A Spanning Tree-Based SAT Encoding for Reachability in Undirected Graphs

Miquel Bofill*, Cristina Borralleras†, Joan Espasa‡ and Mateu Villaret*
*Universitat de Girona, Girona, Spain
†Universitat de Vic - Universitat Central de Catalunya, Vic, Spain
‡University of St Andrews, St Andrews, UK

***Abstract*—The most studied case for reachability in graphs is st-connectivity, where, for a pair of vertices s and t, it is determined if t is reachable from s. We review some existing st-connectivity SAT encodings and discuss their suitability for the case of grids, with special emphasis in Sokoban-like games. We argue that in the context of planning as SAT, an encoding that may be appropriate for sequential plans may not be appropriate at all for parallel plans, and vice versa. Moreover, we propose a new st-connectivity encoding, which outperforms other encodings in the parallel case.**

I. Introduction

The problem of reachability can be understood as the question of whether a vertex can be reached from another in a graph. Given its generality and usefulness, reachability is used in many and varied settings, where it is normally not used alone but as part of a more complex problem. For example, in puzzle video games such as Sokoban ([1], [2]), the difficulty of a level is often measured by the number of times the agent needs to move a reachable object in a grid (Figure 1).

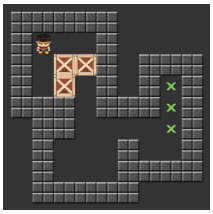


Fig. 1. A Sokoban problem instance.

In this paper, we study the strengths and weaknesses of different propositional encodings of reachability. We take Sokoban as a case study and analyze which reachability encodings are best suited for sequential and parallel plans [3], [4] in a planning as SAT approach for this game. The main contribution is a novel encoding that shows to outperform existing ones in the performed experiments for parallel plans.

The paper proceeds as follows. In Section II we review some existing st-connectivity encodings for graphs, highlighting a simple and efficient encoding for undirected graphs in Subsection II-A. Then, in Section II-B, we present a new encoding, based on spanning trees, which aims to determine the connected component of a vertex in an undirected graph. In Section III, we review the planning as SAT approach to classical planning, concentrating on parallel plans for box-pushing puzzle games with reachability constraints. In Section IV we discuss experimental results and analyze the weaknesses and strengths of the encodings presented, and conclude in Section V.

This work was funded by MCIN/MICIU/AEI/10.13039/501100011033 and by ERDF A way of making Europe (grants PID2021-122274OB-I00 and PID2024-157625OB-I00).

II. st-Connectivity

The problem of st-connectivity consists in determining, for a pair of vertices s and t in a graph, whether t is reachable from s. This problem has been widely studied because of its practical interest. Its complexity, for the restricted case of planar graphs, has been studied in [5], showing that it is complete for nondeterministic logspace (NL). We are interested in declarative approaches to st-connectivity. A thorough study on logic-based characterizations of acyclicity and reachability conditions and their corresponding encodings in the language of answer set programming (ASP) was developed in [6]. A generic ASP encoding for reachability would look like:

```
reachable(T,S) :- S = T.
reachable(T,S) :- reachable(T,S1),
                  adjacent(S1,S).
```

However, the way of characterizing reachability is intrinsically different in SAT than in ASP, the reason being that SAT does not adhere to the closed-world assumption. A direct translation of the previous reachability rules to SAT would give us a formula which could be satisfied by setting all *reachable* variables to true, and hence not encoding reachability at all.

In order to deal with the open-world assumption, in SAT we essentially need to break cyclic relations, i.e., paths from source to target must be encoded as a transitive and antisymmetric relation. As described in [7], acyclicity can be easily modelled with SMT, by imposing an ordering on vertices according to numeric values associated to them. Considering a directed graph $G = (V, E)$ and a source vertex $s \in V$, the encoding goes as follows. For every $v \in V$, a Boolean variable r_v denotes if vertex v is reachable from s. For every edge $(v, v') \in E$, a Boolean variable $e_{vv'}$ indicates if edge (v, v') is in the reachability path. Moreover, to avoid cycles in those paths, an numeric variable $a_v \in 1..|V|$ is introduced for each vertex $v \in V$. Cycles are then forbidden by enforcing a

topological ordering between the vertices in the reachability path. The constraints are the following:

$$r_s \tag{1}$$

$$\forall_{v \in V \setminus \{s\}} \ \big(r_v \rightarrow \bigvee_{(v',v) \in E} e_{v'v}\big) \tag{2}$$

$$\forall_{(v,v') \in E} \ \big(e_{vv'} \rightarrow r_v \wedge a_v < a_{v'}\big) \tag{3}$$

Additionally, a unit clause r_t must be imposed to require reachability from s to the desired vertex t.

Translation of Ordering Constraints to SAT.: The numeric constraints $a_v < a_{v'}$ in (3) can be easily encoded to SAT, either by encoding the numbers in binary form or, preferably, by encoding the acyclicity relation directly to SAT, as explained below.

A *strict partial order* is a relation $<$ that is irreflexive and transitive (which implies antisymmetry as well). This is all we need to ensure acyclicity. We can encode such a relation by adding the constraints $\neg t_{vv}$ (irreflexivity) and $t_{vv'} \wedge t_{v'v''} \rightarrow t_{vv''}$ (transitivity) where $t_{vv'}$ are Boolean variables, for vertices v, v'. Notice that transitivity constraints $t_{vv'} \wedge t_{v'v''} \rightarrow t_{vv''}$ are only needed for neighbours v' of v, since transitivity follows by induction. This encoding has $\mathcal{O}(|V||E|)$ clauses and $\mathcal{O}(|V|^2)$ variables.

A similar encoding was given in [7], based on the transitive closure of the relation corresponding to the underlying graph: variables $t_{vv'}$ indicate that (v, v') is in the transitive closure, variables $e_{vv'}$ for edges (v, v') imply $t_{vv'}$, transitivity is expressed by $e_{vv'} \wedge t_{v'v''} \rightarrow t_{vv''}$, and cycles are forbidden by $e_{vv'} \rightarrow \neg t_{v'v}$. This encoding also has $\mathcal{O}(|V||E|)$ clauses and $\mathcal{O}(|V|^2)$ variables. In either case, after adding these constraints and replacing (3) by $\forall_{(v,v') \in E} \ (e_{vv'} \rightarrow r_v \wedge t_{vv'})$, we obtain a full SAT encoding which has $\mathcal{O}(|V||E|)$ clauses and $\mathcal{O}(|V|^2)$ variables. It is worth noting that this encoding computes a directed acyclic graph covering a subset of the reachable vertices which includes the desired target. For this reason, in the following we refer to this encoding as ***DAG encoding***.

A. A Simple Encoding for Grids

A different approach to connectedness is that of counting-based encodings. A (nonempty) path between s and t in an undirected graph G exists iff there is a subgraph of G in which the degree of both s and t is exactly 1, and the degree of all other vertices is 0 or 2 [8]. In [8] it is suggested to use cardinality operators to count the degree of each vertex in the subgraph, and to transform them to CNF using recursive counters [9]. Later on, [10] proposed representing these cardinality operators as Pseudo-Boolean constraints, and translating them to CNF by creating a Reduced Ordered Binary Decision Diagram, which is a canonical representation of a Boolean function [11].

Following these ideas, a path between s and t in a grid can be modelled as follows: define a Boolean variable for each location denoting if it is included in the path, and impose that (i) s and t are in the path, (ii) each of them has exactly 1 neighbour in the path, and (iii) any location in the path different from s and t has exactly 2 neighbours in the path. It is worth noting that the trick of restricting s and t to have only one neighbour in the path is precisely what avoids a cycle, but the proposed constraints allow for unrelated cycles in addition to the path. Leaving aside search space considerations, superfluous cycles do not matter if we are only interested in reachability. Moreover, disconnected cycles cannot be easily avoided by adding additional constraints. Removing those cycles would require something similar to the existing more complex encodings of st-connectivity.

Note also that the proposed constraints forbid cycling back through a neighbour. For example, the sequence of locations $(1,1),(1,2),(1,3),(2,3),(2,2)$ would not be allowed, since location $(1,2)$ would have 3 of its neighbours in the path (left grid below). Nevertheless, the constraints are enough to capture reachability (as exemplified in the right grid below).

Cardinality constraints can be encoded in many ways. But, since in a grid (with obstacles) each location will have at most 4 neighbours, no sophisticated encoding is needed. In fact, for $n \leq 4$ the binomial encoding of the at-most-k constraint has fewer clauses than other encodings [12]. The at-most-k constraint, written $\leq_k(x_1, \ldots, x_n)$, is true iff at most k of the $x_1, \ldots, x_n$ Boolean variables are true. Its binomial encoding is

$$\bigwedge_{\substack{I \subseteq \{1,\ldots,n\}, \\ |I| = k+1}} \ \bigvee_{i \in I} \neg x_i$$

The encoding introduces no new variables and comprises $\binom{n}{k+1}$ clauses of size $k+1$. For at-least-k, we have $\geq_k(x_1, \ldots, x_n) \equiv \leq_{n-k}(\neg x_1, \ldots, \neg x_n)$. And, for exactly-$k$, $=_k(x_1, \ldots, x_n) \equiv \geq_k(x_1, \ldots, x_n) \wedge \leq_k(x_1, \ldots, x_n)$.

As said, the counting-based encoding of st-connectivity requires a new variable p_l to indicate whether each location l is included in the path. Then, t is reachable from s in a grid if either $s = t$ or there is a nonempty path from s to t according to the following constraints:

$$p_s \tag{4}$$

$$p_t \tag{5}$$

$$=_1 path\_neighbours(s) \tag{6}$$

$$=_1 path\_neighbours(t) \tag{7}$$

$$\forall_{l \in L \setminus \{s,t\}} \ p_l \rightarrow \ =_2 path\_neighbours(l) \tag{8}$$

where L denotes the set of valid locations, i.e., the set of locations without an obstacle, and $path\_neighbours(l) = (p_{l_1}, \ldots, p_{l_n})$ where $l_1, \ldots, l_n$ are the valid neighbours of location l, with $n \leq 4$. For general graphs, this encoding comprises $\mathcal{O}(|V||E|)$ clauses and $O(|V|)$ variables. For the case of grids, since $n \leq 4$ or, more generally, $|E| \leq 4|V|$, we have $\mathcal{O}(|V|)$ clauses and $\mathcal{O}(|V|)$ variables.

Since this encoding computes a path from the source to the target, we refer to it as ***path encoding***.

B. A New SAT Encoding for st-Connectivity in Undirected Graphs

As said, the *DAG* encoding for st-connectivity computes a directed acyclic graph covering a subset of the reachable vertices which includes the desired target. In this section, we present a stronger encoding, which computes a tree rooted at the source vertex that covers all reachable vertices from it, i.e., a spanning tree. This is especially suited for determining the connected component of a vertex, i.e., the set of all vertices that it can reach. The encoding is given for undirected graphs (without self-loops). In the following constraints, the variables r_v denote whether a vertex v is reachable from the source s, and the variables $t_{vv'}$ denote the existence of a path from vertex v to v' in the tree rooted at s.

$$r_s \tag{9}$$

$$\forall_{(v,v')\in E} \quad r_v \rightarrow r_{v'} \tag{10}$$

$$\forall_{v\in V:(s,v)\in E} \quad t_{sv} \tag{11}$$

$$\forall_{v\in V\setminus\{s\}} \quad r_v \rightarrow \bigvee_{(v',v)\in E} t_{v'v} \tag{12}$$

$$\forall_{(v,v')\in E,\,(v,v'')\in E,\,v'\neq v''} \quad \neg t_{v'v} \vee \neg t_{v''v} \tag{13}$$

$$\forall_{(v,v')\in E,\,v''\in V,\,v'\neq v''} \quad t_{vv'} \wedge t_{v'v''} \rightarrow t_{vv''} \tag{14}$$

$$\forall_{v\in V} \quad \neg t_{vv} \tag{15}$$

$$\forall_{(v,v')\in E} \quad t_{vv'} \vee t_{v'v} \rightarrow r_v \tag{16}$$

The explanation is the following: (9) sets the source vertex s as reachable; (10) propagates reachability to neighbours; (11) sets outgoing paths from s to its neighbours; (12) forces at least one path into each reachable vertex, except for s, while (13) forces at most one path into each vertex, so (12) and (13) altogether force exactly one path into each reachable vertex, except for s. Moreover, (14) defines the transitivity of paths, and (15) forbids irreflexivity, and hence cycles. Therefore, (11), (12), (13), (14) and (15) define a tree rooted at s of vertices reachable from s. Moreover, thanks to (9) and (10), that tree will span to all reachable vertices. Finally, (16) sets to reachable any vertex in a path. It is worth noting that the tree of paths defined by the variables $t_{vv'}$ is essential for setting to reachable exactly the vertices that are reachable from the source. The rationale is the following. Since (12) is forcing some path into every reachable vertex except for the source, in the absence of the source reachable vertices would form some cycle. But cycles are forbidden by (15). So at least one vertex must be set to unreachable in areas disconnected from the source. Then, in setting that vertex to unreachable, unreachability spreads out to its neighbours by (10). Note also that (16) is not needed to correctly set the value of the r_v variables, but it forces the $t_{vv'}$ variables to false in disconnected components, thus reducing the search space.

This encoding also has $\mathcal{O}(|V||E|)$ clauses and $\mathcal{O}(|V|^2)$ variables. We refer to it as ***spanning tree encoding***.

III. SAT Encodings for Box-Pushing Puzzle Games

In the planning as SAT approach [13], a planning problem is encoded to a Boolean formula, with the property that any model of this formula corresponds to a valid plan. Since the length of a valid plan is not known a priori, the idea is to encode the existence of a plan of T steps with a formula $f(T)$, and iteratively check the satisfiability of $f(T)$ for $T = 0, 1, 2, \ldots$ until a satisfiable formula is found.

Actions are defined as pairs of preconditions and effects $\langle Pre, \mathit{Eff}\rangle$ (typically, conjunctions of literals). Variables need to be replicated for each time step, e.g., a^t denotes if action a is executed at time t. Then, the formula states that the execution of an action implies its preconditions and effects: for every action $a = \langle Pre, \mathit{Eff}\rangle$ and $t \in 0..T-1$, we have $a^t \rightarrow Pre^t$ and $a^t \rightarrow \mathit{Eff}^{t+1}$, where Pre^t and Eff^{t+1} denote the corresponding formulas on the time-indexed state variables. Moreover, we need to state that a change in the value of a state variable v can occur only if an action that can change this value is executed: for every variable v and $t \in 0..T-1$, we have the frame axioms $v^t \wedge \neg v^{t+1} \rightarrow \bigvee\{a^t \mid a = \langle Pre, \mathit{Eff}\rangle \in A, \neg v \in \mathit{Eff}\}$ and $\neg v^t \wedge v^{t+1} \rightarrow \bigvee\{a^t \mid a = \langle Pre, \mathit{Eff}\rangle \in A, v \in \mathit{Eff}\}$. Finally, it is stated that exactly one action is executed at each time step t, and that the goal holds at time T.

A. Sequential Plans

Here we describe a generic encoding for solving box-pushing puzzle games, following a planning as SAT approach. For clarity and space limitations we do not provide the whole encoding, but the viewpoint (state variables) and an excerpt of the formulas including the goal and relevant transition constraints and frame axioms.

We consider single-agent box-pushing puzzle games, such as Sokoban, where each puzzle consists of a maze with fixed obstacles (walls) and movable obstacles (boxes). In Sokoban, for example, the goal is to push all boxes onto a set of designated storage locations (see Figure 1), but several variations do exist. One of the sources for the difficulty of this kind of games is that many pushes are irreversible, leading to dead-end states from which reaching the goal is impossible.

We represent states with Boolean variables denoting, for each non-wall location, (i) whether there is a movable object in it, and (ii) whether the agent is there. We consider only four actions, corresponding to a movement of the agent in each direction (which may imply pushing a box or not). In other words, we only need to know the direction of movement. We consider L to be the set of valid locations, and T the number of time steps considered. For all $l \in L$ and $t \in 0..T$, we have the following variables: b_l^t (there is a box at location l at time t), and c_l^t (the character is at location l at time t). And for all $t \in 0..T-1$: n^t, s^t, e^t, w^t (direction of action is north, south, east or west). For each time step t in $0..T-1$ we have the following constraints:

- Exactly one action (n^t, s^t, e^t, w^t) is executed.
- Action preconditions and effects (we only give an excerpt of the axioms for the case of moving north).
  Let L_n be the set of valid locations with a wall at north, L_{nn} be the set of valid locations with a wall two steps ahead at north, l_n denote the location at north of a location l, and l_{nn} the location two steps ahead at north

of l. The situations where the agent cannot move north are the following:

$$\forall l \in L_n \quad c_l^t \rightarrow \neg n^t$$
$$\forall l \in L_{nn} \quad c_l^t \wedge b_{l_n}^t \rightarrow \neg n^t$$

If the agent moves, his position changes:

$$\forall l \in L \setminus L_n \quad c_l^t \wedge n^t \rightarrow \neg c_l^{t+1} \wedge c_{l_n}^{t+1}$$

Boxes get pushed:

$$\forall l \in L \setminus (L_n \cup L_{nn}) \quad c_l^t \wedge n^t \wedge b_{l_n}^t \rightarrow \neg b_{l_{nn}}^t \wedge \neg b_{l_n}^{t+1} \wedge b_{l_{nn}}^{t+1}$$

- Frame axioms impose that the state cannot change without a reason, i.e, the character cannot appear or disappear from a location unless it is moved, and a box cannot appear or disappear unless it is pushed. For example,

$$\forall l \in L \setminus L_n \quad \neg c_{l_n}^t \wedge c_{l_n}^{t+1} \wedge n^t \rightarrow c_l^t$$

Reducing Search Space: In order to keep the search space small, we can consider the agent walking next to a box as an atomic action. The encoding presented above can be easily adapted to collapsing actions like this. Removing walking (non-pushing) actions essentially reduces to require the agent being at most at one location at each step, and this location being reachable from the previous one, before each pushing action takes place. Any of the reachability encodings presented in the st-connectivity section can be used with this purpose, considering boxes as additional obstacles.

B. Parallel Plans

Another way of further reducing search space is to consider the execution of several actions at a single time point [14]–[16]. This approach does not model true parallelism, but rather represents one or more sequential plans. It makes SAT-based planning more efficient for two reasons: first, it eliminates the need to consider all possible orderings for most action sets and, second, it reduces the number of time steps required, resulting in smaller formulas.

Here we consider solving box-pushing puzzle games by adhering to the so-called $\forall$-step semantics of parallel plans. This implies that any ordering of the actions at each step in the parallel plan must result in a valid sequential plan. Therefore, interfering actions cannot be scheduled at the same time. We consider two types of interference.

Direct Interference: Since now the problem is reduced to pushing boxes, by direct interference between actions we refer to the case where the affected locations (i.e., the source or destination of the involved boxes) do intersect. Constraints avoiding this kind of interference are straightforward.

Indirect Interference: Since we are omitting walking actions, in the following, by *pushing locations* (or *action locations*) we refer to locations where a pushing action can be performed, i.e., locations next to a box from which they can be pushed.

Let's consider a pair of pushing actions that do not directly interfere, and such that their pushing locations are reachable from the current agent location. By indirect interference between such pair of actions we refer to the case where executing one of them prevents the execution of the other, because of making its pushing location unreachable.

Avoiding this kind of interference is not straightforward. In particular, given a set of (candidate) parallel actions, it is not enough to require all pushing locations to be reachable before and after executing all actions. The following example shows a situation with two actions whose locations are reachable both before and after their parallel execution, but there is no way of sequencing them.

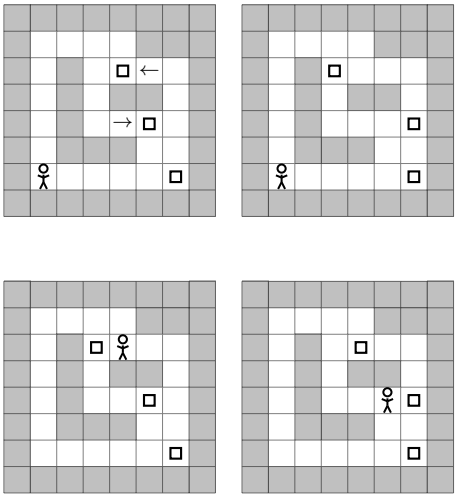


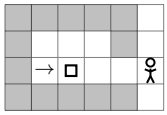
Fig. 2. Indirect interference example.

Example III.1. In Figure 2, greyed squares denote walls, ☥ denotes the current location of the agent, little black squares denote boxes, and arrows denote action locations and the direction of the action (for locations reachable by the agent). Both action locations are reachable by the agent, before and after executing them in parallel (top grids). But this does not imply the actions can be serialized. If we first push the box at the top, we get a situation where the second pushing location becomes unreachable (bottom left grid). If we first push the box at the middle, an analogous situation occurs (bottom right grid).

Therefore, we need to be more restrictive. A sufficient condition for the serializability of parallel actions is to ask for reachability of all action locations considering present and future box locations as occupied. This way, as we are only adding obstacles to the present situation, future reachability of pushing locations is guaranteed. However, this is way too restrictive: although it allows for serialization of parallel actions, it can prevent the execution of some actions, making the encoding incomplete. The following example illustrates this situation.

Example III.2. In the situation below, if we consider the present and future location of the box as occupied, while

asking for reachability from the agent, the box cannot be pushed.



That is, a pushing action can block itself the path used to reach the pushing location.

The solution we propose is to add a new action to jump to a reachable location in a single step. This action will be used exclusively, i.e., not in parallel with any other action. The aforementioned constraints on reachability (considering present and future box locations as occupied) will be imposed only for pushing actions. This way, pushing actions can be executed safely in parallel, while jumping will be performed independently, when needed. It is worth noting that, thanks to the proposed reachability constraints, the agent can stay put during any sequence of pushing actions (i.e., no jumping is necessary).

It is not difficult to see that the proposed system is sound and complete with respect to finding a valid sequential plan from a parallel plan.

IV. Empirical Evaluation

In this section, we evaluate the efficiency of the presented encodings on the Sokoban game. Experiments have been performed on an Intel Xeon E-2234 CPU@3.60GHz, using KISSAT [17] in version 4.0.1. KISSAT and its variations have been the winners of various tracks of the recent SAT competitions [18].[1] We have selected 10 representative instance sets from a large Sokoban repository [19], discarding easy sets with a small number of objects. The selected sets are *bagatelle*, *cantrip*, *cantrip2*, *chessboards*, *dh1*, *GRIGoRusha 2001*, *Sasquatch IX*, *Sharpen*, *SokoStation*, and *Tian Lang*, with a total of 795 instances. Locations in the grids range from 13 to 837, and the number of boxes from 1 to 232.

A. Sequential Plans

We evaluate the performance of the three presented encodings for reachability in grids (*path*, *DAG*, and *spanning tree*), on sequential plans. We aim to check that the smaller *path* encoding works best on sequential plans. Figure 3 shows scatter plots comparing the solving times on the selected instances, when using the three presented encodings for reachability. It can be observed that the *path* encoding outperforms the *DAG* and *spanning tree* encodings in sequential plans, as expected. Recall that the *path* encoding has $\mathcal{O}(N)$ clauses and $\mathcal{O}(N)$ variables, where $N = |V|$ is the number of valid (non-obstacle) locations in the grid. As for the comparison between the *DAG* and the *spanning tree* encodings, the situation is unclear from the scatter plot, although we can appreciate that the *DAG* encoding has more time outs. Recall that both encodings have $\mathcal{O}(|V||E|)$ clauses and $\mathcal{O}(|V|^2)$ variables. Since $|E|$ is $\mathcal{O}(|V|)$ for the case of a grid, these two last encodings have $\mathcal{O}(N^2)$ clauses. Note that all reachability encodings need to be replicated using fresh variables for each time step, since we are considering movable obstacles.

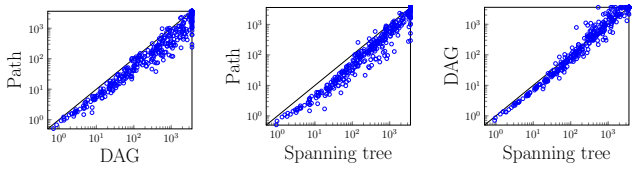


Fig. 3. Comparison of solving times in log scale for the three presented encodings of reachability in Sokoban sequential plans. Time in seconds, limit 1h.

[1] https://satcompetition.github.io/

TABLE I
NUMBER OF SOLVED INSTANCES, TIME OUTS, MEMORY OUTS, AND PAR-2 SCORES FOR SEQUENTIAL PLANNING ON 795 SOKOBAN INSTANCES, DISTINGUISHING BY THE ENCODING USED FOR REACHABILITY.

	Path	DAG	Spanning tree
#solved	**322**	288	299
#TO	**473**	506	493
#MO	**0**	1	3
PAR-2	**3,558,321**	3,814,782	3,742,026

Table I summarizes the results for the 795 instances, where #TO denotes the number of time outs, #MO the number of memory outs, and PAR-2 the sum of all runtimes, taking the double of the time limit for the unsolved instances.

B. Parallel Plans

Here we show that, for parallel plans, the situation is reversed. The *DAG* and *spanning tree* encodings compute a graph of reachable vertices, independently of the target. Therefore, even in the case of performing several actions in parallel, we only need to replicate the encodings for each time step t and ask for reachability of the desired targets (the pushing locations) at time t.

However, the *path* encoding computes a path from the source to a single target location. Therefore, to use it in parallel plans, we need to replicate it for every target, i.e., we need to define as many paths as objects to move. Let M be the number of boxes. Since potentially all of them can be moved simultaneously, we need M path variables p^t_{il} for each valid location l and time step t, with $i \in \{1, \ldots, M\}$, denoting that location l is reachable from the source (the agent location) in path number i at time t. We refer to this encoding as ***multipath encoding***.

Figure 4 compares the solving times on the selected instances, when seeking for a parallel plan using the three encodings for reachability, and table II summarizes the results. We observe that both the *DAG* encoding and the *spanning tree* encoding outperform the *multipath* encoding. Moreover, now the *spanning tree* encoding is clearly better than the *DAG* encoding, especially on hard instances. Recall that the *spanning tree* encoding sets to reachable exactly all reachable locations, by computing a spanning tree rooted at the agent location. The *DAG* encoding is weaker, in the sense that it only

computes a partial digraph, being a spanning tree a particular case. Thus, the *DAG* encoding is going to translate into more models and, in the case of a sequence of unsatisfiable instances to refute, like the ones we face in planning as SAT, it can take longer.

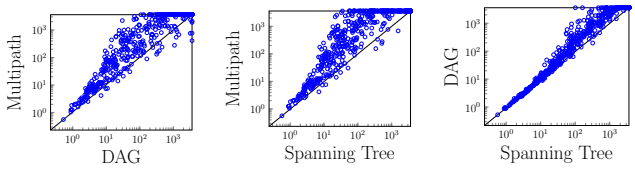


Fig. 4. Comparison of solving times in log scale for the three presented encodings of reachability in Sokoban parallel plans. Time in seconds, limit 1h.

TABLE II
NUMBER OF SOLVED INSTANCES, TIME OUTS, MEMORY OUTS, AND PAR-2 SCORES FOR PARALLEL PLANNING ON 795 SOKOBAN INSTANCES, DISTINGUISHING BY THE ENCODING USED FOR REACHABILITY.

	Multipath	DAG	Spanning tree
#solved	333	440	**499**
#TO	433	354	**295**
#MO	29	1	1
PAR-2	3,517,670	2,774,010	**2,341,613**

Finally, despite the *path* encoding being well suited for a single target, the *multipath* encoding, its counterpart for several targets, shows poor performance in parallel plans.

V. CONCLUSIONS AND FURTHER WORK

We have reviewed some st-connectivity encodings and presented a new one, suitable for determining connected components in undirected graphs. We have compared the performance of the different encodings on grids, using the Sokoban puzzle game as a case study in planning as SAT. We have shown that while counting-based encodings can be appropriate for sequential plans, they are not appropriate at all for parallel ones. Moreover, the newly presented encoding, a variation of a DAG-based classical one, has shown to outperform the classical one in the context of planning as SAT, as it is stronger and so it may allow to refute unsatisfiable instances more easily. The results can be easily exported to other box-pushing puzzle games or to problems where reachability in grids is involved.

As seen, acyclicity is a property closely related to reachability in the context of SAT. Some works have introduced SAT and SMT solvers with support for detecting acyclicity and reachability in graphs [7], [20]. Therefore, an alternative approach could be to use a SAT or SMT solver with built-in reachability support, such as MONOSAT [20].

REFERENCES

[1] J. Culberson, "Sokoban is PSPACE-complete," 97-02, Department of Computer Science, University of Alberta, Tech. Rep., 1997.

[2] N. Zhou and A. Dovier, "A Tabled Prolog Program for Solving Sokoban," *Fundam. Informaticae*, vol. 124, no. 4, pp. 561–575, 2013.

[3] H. A. Kautz and B. Selman, "Pushing the envelope: Planning, propositional logic, and stochastic search," in *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'96, Portland/OR, USA*. AAAI Press, 1996, pp. 1194–1201.

[4] A. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artif. Intell.*, vol. 90, no. 1-2, pp. 281–300, 1997.

[5] E. Allender, D. A. Mix Barrington, T. Chakraborty, S. Datta, and S. Roy, "Planar and grid graph reachability problems," *Theory of Computing Systems*, vol. 45, no. 4, pp. 675–723, 2009.

[6] M. Gebser, T. Janhunen, and J. Rintanen, "Declarative encodings of acyclicity properties," *Journal of Logic and Computation*, vol. 30, no. 4, pp. 923–952, 2020.

[7] ——, "SAT Modulo Graphs: Acyclicity," in *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, 2014, pp. 137–151.

[8] R. Tavares de Oliveira, F. Silva, B. C. Ribas, and M. A. Castilho, "On modeling connectedness in reductions from graph problems to extended satisfiability," in *Advances in Artificial Intelligence – IBERAMIA 2012*, J. Pavón, N. D. Duque-Méndez, and R. Fuentes-Fernández, Eds. Springer Berlin Heidelberg, 2012, pp. 381–391.

[9] D. E. Muller and F. P. Preparata, "Bounds to Complexities of Networks for Sorting and for Switching," *Journal of the ACM*, vol. 22, no. 2, pp. 195–201, 4 1975.

[10] R. T. d. Oliveira and F. Silva, "SAT and MaxSAT Encodings for Trees Applied to the Steiner Tree Problem," in *2014 Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE, 10 2014.

[11] I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez Carbonell, and V. Mayer Eichberger, "A New Look at BDDs for Pseudo-Boolean Constraints," *Journal of Artificial Intelligence Research*, vol. 45, pp. 443–480, 11 2012.

[12] A. M. Frisch and P. A. Giannaros, "SAT Encodings of the At-Most-k Constraint. Some Old, Some New, Some Fast, Some Slow." in *The 9th International Workshop on Constraint Modelling and Reformulation (ModRef 2010)*, 2010.

[13] H. A. Kautz and B. Selman, "Planning as satisfiability," in *10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings*, 1992, pp. 359–363.

[14] J. Rintanen, K. Heljanko, and I. Niemelä, "Planning as Satisfiability: Parallel Plans and Algorithms for Plan Search," *Artificial Intelligence*, vol. 170, no. 12-13, pp. 1031–1080, 2006.

[15] M. Wehrle and J. Rintanen, "Planning as Satisfiability with Relaxed Exists-Step Plans," in *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings*, 2007, pp. 244–253.

[16] T. Balyo, "Relaxing the relaxed exist-step parallel planning semantics," in *25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013*. IEEE Computer Society, 2013, pp. 865–871.

[17] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020," in *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Report Series B, T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2020-1. University of Helsinki, 2020, pp. 51–53.

[18] N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, "SAT competition 2020," *Artificial Intelligence*, vol. 301, p. 103572, 2021.

[19] Sokoban, "Sokoban Repository," http://sokobano.de/en/levels.php, 2023, last accessed 2025/03/19.

[20] S. Bayless, N. Bayless, H. H. Hoos, and A. J. Hu, "SAT Modulo Monotonic Theories," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, B. Bonet and S. Koenig, Eds. AAAI Press, 2015, pp. 3702–3709. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9951