(PARIS) Planning Algorithms for Reconfiguring Independent Sets

Remo Christen[1], Salomé Eriksson[1], Michael Katz[2], Emil Keyder[3], Christian Muise[4], Alice Petrov[4], Florian Pommerening[1], Jendrik Seipp[5], Silvan Sievers[1], and David Speck[6]

[1]University of Basel
[2]IBM T.J. Watson Research Center
[3]Invitae
[4]Queen's University
[5]Linköping University
[6]University of Freiburg

1 Introduction

The general approach to all of the solver tracks was to model the ISR problem as one of automated planning, and use a selection of state-of-the-art solvers to solve these instances. Throughout this document, we describe the encoding, solvers, and overall search setup.

2 Planning Encoding

There are two main encodings we considered – *single* and *split*. The former uses a single action to move a token from one location to another, while the latter uses two actions – one to pick up a token and another to place it. While the full details on the Planning Domain Definition Language (PDDL) is out of scope for this document, the snippets presented here are fairly self-explanatory. More details on the PDDL standard can be found in [Haslum et al., 2019].

Figure 1 shows the PDDL code for a single action move, with comments interleaved. Rather than define the actions in a lifted manner, we found that generating the ground actions (along with the conditions necessary to retain the independent set) was the most effective.

Figure 2 shows the pair of actions for the split encoding. Ultimately, we found this encoding to be the most effective, and so was used for all tracks and solvers.

```
(:action move-l7-l5

    :precondition
      (and
        ; Destination free
        (free l5)

        ; Source has token
        (tokened l7)

        ; Destination's neighbours are free
        (free l3) (free l6))

    :effect (and

        ; Move the token from l7 to l5
        (not (tokened l7))
        (free l7)
        (not (free l5))
        (tokened l5))
)
```

Figure 1: PDDL example of a single action move.

```
(:action pick-l7
    :precondition
      (and
        ; Token at the location
        (tokened l7)
        ; Not holding one currently
        (handfree))
    :effect
      (and
        ; Token no longer there
        (not (tokened l7))
        (free l7)
        ; Now holding a token
        (not (handfree))
        (holding))
)
```

```
(:action place-l7
    :precondition
      (and
        ; Holding a token
        (holding)
        ; Destination is free
        (free l7)
        ; Destination neighbours are free
        (free l2))
    :effect
      (and
        ; No longer holding a token
        (not (holding))
        (handfree)
        ; Destination contains the token
        (tokened l7)
        (not (free l7)))
)
```

Figure 2: PDDL example of a pair of split actions for moving a token.

Finally, the planning systems we used are all built on a common software framework which first parses and preprocesses the PDDL into an intermediate form known as SAS+ [Bäckström and Nebel, 1995]. To save on this computational effort, we instead directly encoded the input problem instances for the ISR contest into the SAS+ format. While in general this may lead to a degradation in performance (some planning problems benefit greatly from the preprocessing), in the ISR setting it was far more effective to skip this initial phase of planner technology.

3 Engine: Core Solver or Algorithm

We use sequential algorithm portfolios for each of the three solver tracks. That is, we run a sequence of algorithms, each with an associated time limit. The next section describes the algorithms that we use in our sequential portfolios.

3.1 Our algorithms

A*+Landmarks We run an A* search [Hart et al., 1968] with a *landmark count* heuristic [Karpas and Domshlak, 2009] that uses two different kinds of landmarks: h^1 landmarks [Keyder et al., 2010] and RHW landmarks [Richter et al., 2008]. The landmark costs are combined with *uniform cost partitioning* [Katz and Domshlak, 2008].

GBFS+Landmarks We run a greedy best-first search (GBFS) [Doran and Michie, 1966] with a *landmark count* heuristic [Keyder et al., 2010] that computes all landmarks of the *delete-relaxed* task [Bonet and Geffner, 2001] (h^1 landmarks). The landmark costs are combined with *uniform cost partitioning* [Katz and Domshlak, 2008].

Symbolic search We run a forward symbolic blind search [Torralba et al., 2017, Speck et al., 2020a] using Binary Decision Diagrams [Bryant, 1986] as the underlying data structure. The symbolic planner we use is SymK [Speck et al., 2020b], which uses CUDD [Somenzi, 2015] as its decision diagram library. This algorithm is optimal, sound, and complete, i.e., if it reports a plan, this is a shortest plan, if it reports unsolvability, the task is indeed unsolvable, and given sufficient resources, it will eventually find a shortest plan.

Symbolic top-k search We run a modified forward symbolic blind search based on an algorithm called SymK-LL [von Tschammer et al., 2022], implemented in the symbolic planner SymK [Speck et al., 2020b], which iteratively finds and generates all loopless plans of a given task. However, we have made the following two major adjustments to solve the problem of finding the longest loopless plan feasible. First, once we find a goal state $s_\star$ reachable with a certain cost c, we reconstruct only one loopless plan with cost c and ignore all other plans leading to $s_\star$ or any other goal state reachable with c. Second, since the

split encoding introduced intermediate states in which a token is picked up, we ignore these artificial states when evaluating whether a plan is loopless during the plan reconstruction of SymK-LL. This algorithm has the interesting property that it iteratively finds longer plans, starting with the shortest one, and eventually finds the longest loopless plan if enough resources are available.

Numeric abstraction We abstract the problem to a numeric planning problem and check for unsolvability in the abstraction. Since this algorithm is new, we describe it in more detail in Section 4.

We now describe our sequential algorithm portfolios. Our portfolio for the *existent* track is identical to the one for the *shortest* track.

3.2 Portfolio for *shortest* and *existent* tracks

We list the algorithms and their time limits (the first to find a solution is saved and the rest of the steps ignored):

1. Numeric abstraction: 10sec
2. Symbolic search: 70min
3. A*+Landmarks: 70min
4. GBFS+Landmarks: 70min
5. Numeric abstraction: 14hr

3.3 Single best solver for *shortest* and *existent* tracks

The following single solver was used as a single-solver submission. It had the best coverage among all solvers in the portfolio.

- GBFS+Landmarks: 70min

3.4 Portfolio for *longest* track

We list the algorithms and their time limits:

1. GBFS+Landmarks: 330 seconds
2. Symbolic top-k search: 70min

As a fall-back, if neither of the above approaches produced a solution longer than the shortest/existent track, we used the solution to the shortest/existent track as a default.

3.5 Single best solver for *longest* track

The following single solver was used as a single-solver submission. It had the best coverage among all solvers in the portfolio. Note that we did not use the "fallback" option for this single track submission.

- Symbolic top-k search: 70min

4 Numeric Abstraction

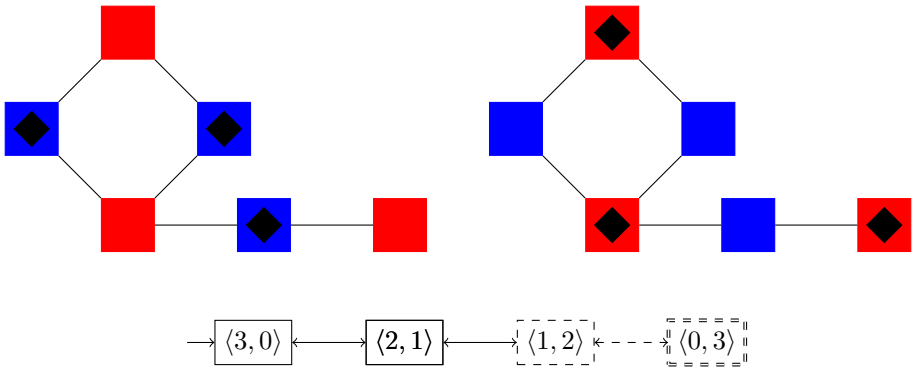

Figure 3: Example coloring for numeric abstraction. The initial state is on the left and the goal state on the right. Nodes are colored blue if they have a token in the initial state but not the goal state and red if they have no token in the initial state but a token in the goal state. The abstract state space is shown below the instance. Dashed nodes are pruned.

The *numeric abstraction* component of our solver tries to detect if the task is unsolvable by abstracting it to a numeric planning problem. Given an ISR problem, we come up with a *coloring* of the vertices in the graph, i.e., a function that maps each vertex of the graph to one color. Many different ways of coming up with a good coloring are conceivable but here we opted for a simple strategy that uses up to four colors:

- one for nodes that contain a token both in the initial state and in the goal state;
- one for nodes that contain a token only in the initial state but not in the goal state;
- one for nodes that contain a token only in the goal state but not in the initial state; and
- one for nodes that are empty in the initial state and goal state.

Colors for situations that do not occur are not used. For example, the task in Figure 3 only requires two colors with the method above.

Given a coloring, each configuration of tokens can be abstracted to a state with one numeric variable per color that counts how many token currently are on vertices with this color. For example, the initial state in Figure 3 has 3 tokens on blue nodes and 0 on red nodes, so it can be represented as the numeric state $\langle 3, 0\rangle$. The goal state has all three tokens on red nodes and none on blue, so it can represented by $\langle 0, 3\rangle$.

Moving any token from a node colored c_i to a node colored c_j changes the abstract state from $\langle c_1, \ldots, c_i, \ldots, c_j, \ldots, c_n\rangle$ to $\langle c_1, \ldots, c_i{-}1, \ldots, c_j{+}1, \ldots, c_n\rangle$. The main observation for this component is that if any solution to the full problem exists, there has to be a solution in our abstraction as well. We thus construct the state space of the abstraction in the following way.

For a state $s = \langle c_1, \ldots, c_n\rangle$, we construct one successor for each pair of different colors c_i and c_j that differs from s by a single token that moved from c_i to c_j. In our running example, the initial state $\langle 3, 0\rangle$ has a single successor $\langle 2, 1\rangle$, and this state has two successors $\langle 3, 0\rangle$ (which we skip because we have already seen this state) and $\langle 1, 2\rangle$. The latter state now has the abstract goal state $\langle 0, 3\rangle$ as a successor (see Figure 3).

Whenever we generate a state, we check whether such a state is possible (independent of whether we are actually able to reach such a state). If it is not possible to place the token on the respective colors in the way suggest, then we do not have to consider it or its successors. In our running example, the state $\langle 1, 2\rangle$ is not realizable: not matter where we place the blue token, it blocks two of the three red nodes. We use a MIP solver to check if a state s is realizable by checking if the following system of constraints have a solution:

$$
\begin{aligned}
x_i + x_j &\leq 1 \quad \text{for all edges } \langle i, j\rangle \text{ in the graph} \\
\sum_{i \in N_c} x_i &\geq s[c] \quad \text{for all colors } c \\
x_i &\in \{0, 1\} \qquad \text{for all nodes } i
\end{aligned}
$$

where N_c is a set of all nodes with color c and $s[c]$ is the amount of tokens that should be on color c in state s. The abstract state s is realizable iff the constraints have a solution.

If we generate a state that matches the goal state ($\langle 0, 3\rangle$ in our example), we know that there is an abstract path to the goal state. In this case, we still do not know if there is a real path to the goal state and return `unknown` (this component of the solver is incomplete). However, if we expand the full abstract state space without finding a path to an abstract goal state, there is no solution to the abstract problem which also means there cannot be a solution to the original problem. The abstract state space is usually small. In our running example, it only has 4 states and we only have to explore 3 of them, as we prune state $\langle 1, 2\rangle$ before reaching a goal state.

We use this component in two places: first with a small time limit at the start of the solver to handle all cases where we can quickly prove unsolvability. Then again with a large time limit after all other components to catch unsolvable instances that are hard to prove unsolvable.

5 Computation Environment

All evaluations were conducted on a single machine running Ubuntu 20.04 (evaluations done through the Docker image for the submission), with the following specs:

- **CPU**: Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz
- **Cores**: 16
- **MEM**: 128Gb

References

C. Bäckström and B. Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

B. Bonet and H. Geffner. Planning as heuristic search. *AIJ*, 129(1):5–33, 2001.

R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

J. E. Doran and D. Michie. Experiments with the graph traverser program. *Proceedings of the Royal Society A*, 294:235–259, 1966.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool, 2019. ISBN 9781627058759. URL `http://www.morganclaypoolpublishers.com/catalog_Orig/product_info.php?products_id=1384`.

E. Karpas and C. Domshlak. Cost-optimal planning with landmarks. In *Proc. IJCAI 2009*, pages 1728–1733, 2009.

M. Katz and C. Domshlak. Structural patterns heuristics via fork decomposition. In *Proc. ICAPS 2008*, pages 182–189, 2008.

E. Keyder, S. Richter, and M. Helmert. Sound and complete landmarks for and/or graphs. In *Proc. ECAI 2010*, pages 335–340, 2010.

S. Richter, M. Helmert, and M. Westphal. Landmarks revisited. In *Proc. AAAI 2008*, pages 975–982, 2008.

F. Somenzi. CUDD: CU decision diagram package - release 3.0.0. https://github.com/ivmai/cudd, 2015. Accessed: 2022-03-24.

D. Speck, F. Geißer, and R. Mattmüller. When perfect is not good enough: On the search behaviour of symbolic heuristic search. In *Proc. ICAPS 2020*, pages 263–271, 2020a.

D. Speck, R. Mattmüller, and B. Nebel. Symbolic top-k planning. In *Proc. AAAI 2020*, pages 9967–9974, 2020b.

Á. Torralba, V. Alcázar, P. Kissmann, and S. Edelkamp. Efficient symbolic search for cost-optimal planning. *AIJ*, 242:52–79, 2017.

J. von Tschammer, R. Mattmüller, and D. Speck. Loopless top-k planning. In *Proc. ICAPS 2022*, 2022.