On the Complexity of STRIPS$^1_1$

Stefan Edelkamp
Department of Theoretical Computer Science
and Mathematical Logic
Faculty of Mathematics and Physics
Charles University in Prague, Czech Republic
edelkamp@ktiml.mff.cuni.cz

Anders Jonsson
Artificial Intelligence and Machine Learning Group
Departament de Tecnologies de la Informaci
i les Comunicacions
Pompeu Fabra University, Barcelona, Spain
anders.jonsson@upf.edu

Abstract

This paper works on top of Bylander's (1994) results on considering the computational complexity of propositional STRIPS planning. Different planning problems can be defined by restricting the type of formulas, placing limits on the number of pre- and postconditions. He showed that when only ground literals are permitted, determining plan existence is PSPACE-complete even if operators are limited to two preconditions and two postconditions. There is an open debate to the complexity theoretical classification of STRIPS$^1_1$, namely whether or not propositional STRIPS with operators that only have one preconditionand one effect, is NP-complete or generates exponentially long plans. We give new theoretical and empirical insights to this amazing yet unsolved problem.

Introduction

STRIPS (Fikes and Nilsson 1971), the acronym of the Stanford Research Institute Planning System, is one the earliest formalisms for action planning. The STRIPS planning language has greatly influenced later developments of the planning domain description language PDDL as proposed by McDermott (2000), Fox and Long (2003) or Hoffmann and Edelkamp (2006), and has been compared to other finite-state variable planning such as SAS$^+$ (Bäckström and Nebel 1993; Nebel 1999; Helmert 2006).

Bylander (1994) studied the complexity of STRIPS planning and showed that the problem of determining plan existence is PSPACE-complete, meaning that the decision problem can be solved using polynomial memory, and that every PSPACE-complete problem polynomially reduces to it. Bylander's reduction proof uses an encoding of a Turing machine with STRIPS operators. PSPACE containment is derived through a divide-and-conquer planning strategy.

An important research question is to establish fragments of STRIPS planning that have lower complexity, i.e. polynomial or NP-complete. Even though it is at present unknown whether or not P $\neq$ NP (arguably the most famous unsolved problem in computer science), it appears highly unlikely that the polynomial hierarchy collapses, in which case such fragments are indeed easier to solve than general STRIPS planning. Even when not directly applicable to a given STRIPS planning problem, tractable fragments are commonly used to compute heuristics that guide search (Hoffmann and Nebel 2001; Katz and Domshlak 2008).

One famous fragment of STRIPS planning is the delete relaxation (Hoffmann and Nebel 2001), which is known to be NP-complete. Bylander took a different route and asked what happens if the operators of a STRIPS problem is limited to at most p preconditions and e effects, giving rise to fragments STRIPS^e_p. Concretely, Bylander showed that STRIPS$^1_1$ is NP-hard, and later Jonsson et al. (2014) proved that STRIPS$^1_2$ is PSPACE-complete[1]. However, the exact complexity of STRIPS$^1_1$ remains unknown, i.e. there exists no proof of containment in NP nor for PSPACE-hardness.

In this paper, we consider the following question for STRIPS$^1_1$: Given a set of propositions (facts, atoms) $F = \{f_1, \ldots, f_n\}$, is it possible to find a set of STRIPS actions $A = a_1, \ldots, a_m$ of at most one precondition and one effect that generate shortest plans of exponential size in n?

This is an open research problem, and this paper attempts to shine some new light on this question. The implications are as follows:

- If there are exponentially long shortest plans, we would need exponential time generating them. While such EXPTIME problems do not contradict polynomial space (as it was shown with the general STRIPS problem), by definition we could get closer to the assertion that the problem was not in NP. However, from an exponential lower bound on the plan length, we cannot directly infer that the problem was not in NP, as there might be other polynomial certificates.
- If at least one plan has polynomial length, then we can verify it in polynomial time, implying containment in NP.

We start with a mapping of the problem to a hypercube and a first example. We formally define and present a STRIPS$^1_1$ planning task that provably produces polynomially long optimal plans, followed by further empirical findings.

[1] In Chapter 5 they write *Bylander showed that the problem of plan existence is PSPACE-complete for STRIPS planning instances whose actions have one postcondition and unbounded number of preconditions (either positive or negative). He did not prove a result for a bounded number k of preconditions, but conjectured that plan existence falls into the polynomial hierarchy in a regular way, with the precise complexity determined by k. In this section we modify our previous reduction such that actions have at most two preconditions, thus showing that plan existence is PSPACE-complete for $k = 2$ and proving Bylander's conjecture to be wrong.*

Hypercubes and A Motivating Example

To arrive at some assessment about the status of STRIPS$_1^1$, let us first take a look at the n-dimensional hypercube over $\{0,1\}^n$ with planning states being mapped to hypercube nodes and STRIPS$_1^1$ actions with a single effect follow one edge. The hypercube on n variables has 2^n nodes and $n2^{n-1}$ undirected edges, or $n2^n$ directed edges. We have $2n \cdot 2n = 4n^2$ different STRIPS$_1^1$ actions. All connect about the same number of nodes. For each pair of adjacent nodes, there are n preconditions that hold, i.e. each edge corresponds to n actions. If we color the hyperedges using n different colors, each action *represents* $n^2 2^n / 4n^2 = 2^{n-2}$ directed colored edges, which is an exponential number, quite substantial for a graph with 2^n nodes. This makes it hard to form exponentially long plans with no cycles, since each STRIPS$_1^1$ action covers 2^{n-2} edges, or 2^{n-1} for precondition-free actions.

Let us consider an example of a STRIPS$_1^1$ planning problem with $n = 4$ variables named $0, 1, 2, 3$. We use $npre(x)$ and $pre(x)$ to denote negative and positive preconditions on x, and we use $add(x)$ and $del(x)$ to denote add and delete effects on x. The initial state is 0000, the goal state is 1011, and the action set is defined as follows:

$$
\begin{array}{lcl}
a_1 & : & npre(2), add(0) \\
a_2 & : & npre(3), add(0) \\
a_3 & : & pre(3), del(0) \\
a_4 & : & pre(0), add(1) \\
a_5 & : & npre(0), del(1) \\
a_6 & : & pre(1), add(2) \\
a_7 & : & npre(1), del(2) \\
a_8 & : & pre(2), add(3) \\
a_9 & : & npre(2), del(3)
\end{array}
$$

The shortest plan is

$$a_1, a_4, a_6, a_8, a_3, a_5, a_7, a_1, a_4, a_3, a_9, a_6, a_5, a_2, a_8$$

with corresponding state sequence 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001,1001, 1101,0101, 0100, 0110, 0010, 1010, 1011. As the example shows, some actions are applied more than once. However, no two states appear more than once, as the plan would no longer be the shortest.

The question is: *how to select the set of actions that has the longest of all shortest path on the induced graph?*

For 5 variables it is already challenging to enumerate all problems in STRIPS$_1^1$ (even if symmetries are exploited). We cannot prove that any problem with 5 variables has the shortest solution of 31 (the longest minimal solution we have found has a length of 29). It is, however, easy to see that when the causal graph is acyclic, all problems have a plan length bounded by $n(n-1)/2$. For the general case, this remains an open problem.

Preliminaries and Exploration

In this work we focus on the fragment STRIPS$_1^1$:

Definition 1 *(STRIPS$_1^1$) We define STRIPS$_1^1$ to be the decision problem of finding a plan to a planning task with n propositional atoms $\mathcal{F}$, initial state $\mathcal{I} \subseteq F$, goal condition $\mathcal{I} \subseteq F$, a set $\mathcal{O}$ of operators $o = (npre, pre, add, del)$, $o \in \mathcal{O}$ with $npre, pre, add, del \subseteq F$ and $|npre \cup pre| = |add \cup del| = 1$.*

The upper bound on the state space of any propositional STRIPS problem is at most 2^n. We cannot assume an optimal plan to be longer as it would contain a self-loop.

We know that plan existence in STRIPS$_1^1$ is NP-hard (as shown by Bylander) and in PSPACE (like any STRIPS problem). To prove that the class is in NP we would have to show that all planning problems in the class have *some* polynomial-length solution. Note that preconditions can also be negative (with only positive preconditions the problem of plan existence is in P, cf. Theorem 3.7 of Bylander).

First of all, we identify the propositional atoms $f_1, \ldots, f_n$ by their index values $1, \ldots, n$, and apply an index shift to obtain $0, \ldots, n-1$. We use a hypercube representation of the state space. Each set of facts corresponds to a bit vector in $\{0,1\}^n$, and a binary number in $\{0, \ldots, 2^n - 1\}$.

The problem we are facing is related to the snake-in-the-box problem; a longest path problem in a n-dimensional hypercube. The design of a long snake has impact for the generation of improved error-correcting codes. The snake increases in length, but must not approach any of its previous visited vertices with Hamming distance 1 or less. It has be proven that the snake-in-the-box problem generates longest plans that are exponentially large (Abbott and Katchalski 1988; 1991). The formal definition of the problem and its variants as well as heuristic search techniques for solving it have been studied by (Palombo et al. 2015; Edelkamp and Cazenave 2016). Some state-of-the-art results were found with variants of NRPA (Dang et al. 2023).

For a growing number n we conducted an exhaustive search on generating actions for the STRIPS$_1^1$ planning task for which the smallest plan is as large as possible. To allow for fast operation and hashing, we encode all states as numbers in $\{0, \ldots 2^n - 1\}$, so that the binary representation can be accessed to reflect the facts that are true and false.

The recursive generator (see pseudo-code in Figure 2) for STRIPS$_1^1$ problems works as follows. It reduces the number of actions to be produced for the domain given as a parameter. Once the parameter approaches 0, it initializes the hash table and calls the solver. The maximum shortest distance is traced. For each action we loop on all possible precondition and effect. To cope with the large state spaces, we apply pruning rules for the actions chosen. Duplicate that lead to equivalent rules are also removed. We used breadth-first search (BFS) to determine the length of the shortest plan for the planning task generated (see Figure 3). It features a queue and a perfect hash table.

A STRIPS$_1^1$ Scheme

One schematic way to generate STRIPS$_1^1$ problems is the following:

Definition 2 *(STRIPS$_1^1$ Planning Task $\mathcal{P}_n$) For each n, we introduce the following actions:*

- $pre(i-1), add(i)$ *(add i whenever $i-1$ is true)*
- $npre(i-1), del(i)$ *(delete i whenever $i-1$ is false)*

Let $\mathcal{P}_n$ be the STRIPS$^1_1$ planning task including the above actions, and the following actions:

- $npre(n-1), add(0)$ *(add* 0 *when* $n-1$ *is false)*
- $npre(2), add(0)$ *(add* 0 *when* 2 *is false)*
- $pre(n-1), del(0)$ *(delete* 0 *when* $n-1$ *is true)*

The initial state is always $0 \cdots 0$. We established that $\mathcal{P}_n$ generates long minimal plans, so it provides a useful starting point for exploration. However, as we shall see, this scheme does not give the optimal solution for all n.

Our program generates the following planning task $\mathcal{P}_6$:

$$
\begin{aligned}
a_0 &: pre(0), add(1)\\
a_1 &: pre(1), add(2)\\
a_2 &: pre(2), add(3)\\
a_3 &: pre(3), add(4)\\
a_4 &: pre(4), add(5)\\
a_5 &: npre(5), add(0)\\
a_6 &: npre(0), del(1)\\
a_7 &: npre(1), del(2)\\
a_8 &: npre(2), del(3)\\
a_9 &: npre(3), del(4)\\
a_{10} &: npre(4), del(5)\\
a_{11} &: pre(5), del(0)\\
a_{12} &: npre(2), add(0),
\end{aligned}
$$

with minimal solution plan (reverse order): $101101 \leftarrow 101001 \leftarrow 101011 \leftarrow 101010 \leftarrow 001010 \leftarrow 011010 \leftarrow 010010 \leftarrow 010110 \leftarrow 010100 \leftarrow 010101 \leftarrow 110101 \leftarrow 100101 \leftarrow 000101 \leftarrow 001101 \leftarrow 011101 \leftarrow 111101 \leftarrow 111001 \leftarrow 110001 \leftarrow 100001 \leftarrow 000001 \leftarrow 000011 \leftarrow 000111 \leftarrow 001111 \leftarrow 011111 \leftarrow 111111 \leftarrow 111110 \leftarrow 111100 \leftarrow 111000 \leftarrow 110000 \leftarrow 100000 \leftarrow 000000.$
Our program found the goal in depth 30 with 64 states being expanded.

For the following planning task $\mathcal{P}_7$

$$
\begin{aligned}
a_0 &: pre(0), add(1)\\
a_1 &: pre(1), add(2)\\
a_2 &: pre(2), add(3)\\
a_3 &: pre(3), add(4)\\
a_4 &: pre(4), add(5)\\
a_5 &: pre(5), add(6)\\
a_6 &: npre(6), add(0)\\
a_7 &: npre(0), del(1)\\
a_8 &: npre(1), del(2)\\
a_9 &: npre(2), del(3)\\
a_{10} &: npre(3), del(4)\\
a_{11} &: npre(4), del(5)\\
a_{12} &: npre(5), del(6)\\
a_{13} &: pre(6), del(0)\\
a_{14} &: npre(2), add(0)
\end{aligned}
$$

we generate the shortest solution plan (reverse order): $1011101 \leftarrow 1011001 \leftarrow 1011011 \leftarrow 1010011 \leftarrow 1010111 \leftarrow 1010110 \leftarrow 0010110 \leftarrow 0010100 \leftarrow 0010101 \leftarrow 0110101 \leftarrow 1110101 \leftarrow 1100101 \leftarrow 1000101 \leftarrow 0000101 \leftarrow 0001101 \leftarrow 0011101 \leftarrow 0111101 \leftarrow 1111101 \leftarrow 1111001 \leftarrow 1110001 \leftarrow 1100001 \leftarrow 1000001 \leftarrow 0000001 \leftarrow 0000011 \leftarrow 0000111 \leftarrow 0001111 \leftarrow 0011111 \leftarrow 0111111 \leftarrow 1111111 \leftarrow 1111110 \leftarrow 1111100 \leftarrow 1111000 \leftarrow 1110000 \leftarrow 1100000 \leftarrow 1000000 \leftarrow 0000000$
We found the goal in depth 35 with 126 states being expanded.

Generating shortest plans we find end states of type $101101010\ldots10101$, for $n = 2k$ and states of type $1011101010\ldots10101$ for n odd, i.e., $n = 2k+1$. We can express them as regular string expressions as follows: $(10)(11)(10)^{k-2}$ for $n = 2k$ and states of $(10)(111)(10)^{k-2}$ for $n = 2k+1$.

For small values of n we see also corner cases, where this construction fails:

- n odd, $k = 2$ so that $n = 2k+1 = 5$. For this case we have $(10)(111)(10)^0 = 10111$ and the precondition $-2, 0$ is affected by 111 pattern;
- n even, $k = 2$ so that $n = 2k = 4$ For this case we have $(10)(11)(10)^0 = 1011$ and the precondition $-2, 0$ is not affected by 11 pattern.

This suggests an induction should be started for $k = 3$, $n = 6$. The growth in the solution length is shown in Table 1.

To compute the regressional growth, we aim at a closed form for the minimal solution length $l(k)$.

Theorem 1 *(Complexity $\mathcal{P}_n$) The empirically validated growth of the optimum solution lengths of $\mathcal{P}_n$ is at most quadratic.*

Proof: For the difference $d(k) = l(k) - l(k-1)$, we have two cases:

- For n being even we obtain the sequence $d_e(k) = +19, +23, +27, +31, +35, \ldots$ (increase is 4);
- For n being odd we obtain the sequence $d_o(k) = +21, +25, +29, +34, +38\ldots$ (increase is 4).

In both cases $d(k) - d(k-1) = 4$, which implies $d(k) = 4 + d(k-1) = ... = 4 \cdot k + c$. In order to determine constant c, we distinguish two different base cases:

k	3	3	4	4	5	5	6	6	7	7	8
n	6	7	8	9	10	11	12	13	14	15	16
l	30	35	49	56	72	81	99	110	130	143	165

Table 1: Growth of optimal solution length l wrt. n and k for $\mathcal{P}_n$.

n	8	9	10	11	12	13	14	15	16
l	49	56	72	81	99	110	130	143	165
e	256	510	1024	2046	4096	2^{13}-2	2^{14}	2^{15}-2	2^{16}
t	0.005	0.009	0.019	0.018	0.031	0.056	0.103	0.189	0.385

n	17	18	19	20	21	22	23	24	25
l	180	204	221	247	266	294	315	345	368
e	2^{17}-2	2^{18}	2^{19}-2	2^{20}	2^{21}-2	2^{22}	o.o.m.	o.o.m.	o.o.m.
t	0.95	1.73	4.28	9.52	19.1	44.1	–	–	–

Table 2: Theoretical and experimental growth in optimal solution lengths for $\mathcal{P}_n$, number of expanded nodes and CPU time in seconds (o.o.m. stands for *out of memory / Java heap space*) wrt. problem size n.

- For n even $d(3) = 19 = 12 + c_e$ implies $c_e = 7$
- For n odd: $d(3) = 21 = 12 + c_o$ implies $c_o = 9$

We have $l(k) - l(k-1) = d(k-1)$, so that $l(k) = l(k-1) + d(k-1)$. Knowing a closed form for d yields

$$\begin{aligned} l(k) &= l(k-1) + (4k + c) \\ l(k) &= l(k-2) + (4k + c) + (4(k-1) + c) \\ &= b + ck + 4(k \sum_{i=0}^{k} i) \\ &= b + ck + 2(k(k-1)) \end{aligned}$$

With $k = n/2$ we also have $l(n) = \Theta(n^2)$

As a result we have shown that the growth of the optimum solution to $\mathcal{P}_n$ is at most quadratic. □

Sharp bounds are derived as follows. Again we distinguish the two cases:

- n even, i.e., $n = 2k$, so that $l(3) = 30 = b_e + 7 \cdot 3 + 2 \cdot 6$ yields $b_e = 30 - 21 - 12 = -3$ and $l(k) = -3 + 5k + 2k^2$;
- n odd, i.e., $n = 2k+1$, so that $l(3) = 35 = b_o + 9 \cdot 3 + 2 \cdot 6$ yields $b_o = 35 - 27 - 12 = -4$ and $l(k) = -4 + 7k + 2k^2$.

We implemented the algorithms in Java and tested them on one core of an Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz with 16GB. Running time results are given in Table 2. It is a simple exercise to validate the formulas, by comparing the values with the ones given in Table 1, we used the formula to determine the minimal solution length values in Table 2 when we ran out of Java heap space.

Progress

We used the STRIPS$^1_1$ problem $\mathcal{P}_n$ as a starting point to search for longer shortest solutions. Concretely, we assumed that some elements of $\mathcal{P}_n$ remained constant, and only iterated over the remaining elements. This way we were able to find problems with longer solutions.

For $n = 5$ the following problem has a shortest solution of length 29. We are confident that this is the longest shortest solution for any STRIPS$^1_1$ problem with 5 variables:

$$\begin{array}{rcl} a_0 & : & pre(1), add(0) \\ a_1 & : & npre(4), add(0) \\ a_2 & : & npre(1), del(0) \\ a_3 & : & pre(0), add(1) \\ a_4 & : & pre(2), add(1) \\ a_5 & : & pre(3), add(1) \\ a_6 & : & pre(4), add(1) \\ a_7 & : & npre(0), del(1) \\ a_8 & : & npre(2), del(1) \\ a_9 & : & pre(1), add(2) \\ a_{10} & : & npre(1), del(2) \\ a_{11} & : & npre(3), del(2) \\ a_{12} & : & pre(2), add(3) \\ a_{13} & : & pre(4), del(3) \\ a_{14} & : & pre(3), add(4) \\ a_{15} & : & pre(0), del(4) \end{array}$$

$$\begin{array}{rcl} a_0 & : & pre(1), add(0) \quad npre(4), add(0) \quad pre(2), add(0) \\ a_1 & : & npre(5), add(0) \quad npre(1), del(0) \quad npre(5), add(0) \\ a_2 & : & npre(1), del(0) \quad pre(0), add(1) \quad npre(1), del(0) \\ a_3 & : & pre(0), add(1) \quad pre(5), add(1) \quad pre(0), add(1) \\ a_4 & : & pre(3), add(1) \quad npre(2), del(1) \quad pre(3), add(1) \\ a_5 & : & pre(4), add(1) \quad pre(1), add(2) \quad pre(4), add(1) \\ a_6 & : & pre(5), add(1) \quad pre(3), add(2) \quad pre(5), add(1) \\ a_7 & : & npre(0), del(1) \quad npre(3), del(2) \quad npre(2), del(1) \\ a_8 & : & npre(2), del(1) \quad pre(2), add(3) \quad pre(1), add(2) \\ a_9 & : & pre(1), add(2) \quad pre(4), add(3) \quad npre(0), del(2) \\ a_{10} & : & npre(3), del(2) \quad npre(0), del(3) \quad npre(3), del(2) \\ a_{11} & : & pre(2), add(3) \quad npre(4), del(3) \quad pre(2), add(3) \\ a_{12} & : & npre(1), del(3) \quad pre(3), add(4) \quad npre(1), del(3) \\ a_{13} & : & npre(4), del(3) \quad pre(0), del(4) \quad npre(4), del(3) \\ a_{14} & : & pre(3), add(4) \quad npre(3), del(4) \quad pre(3), add(4) \\ a_{15} & : & pre(5), del(4) \quad npre(5), del(4) \quad pre(5), del(4) \\ a_{16} & : & pre(4), add(5) \quad pre(4), add(5) \quad pre(4), add(5) \\ a_{17} & : & pre(0), del(5) \quad pre(0), del(5) \quad pre(0), del(5) \end{array}$$

Figure 1: Three systems for $n = 6$ with solution lengths 35, 36, and 38, respectively.

For $n = 6$, the situation is less clear, since search is slow, and it is possible that we are nowhere near the longest solution. These best three solutions minimal plans we found had length 35, 36, and 38, respectively (see Fig 1). If we compare the plans for $n = 5$ and $n = 6$, there are considerable similarities.

Another important finding for $n = 6$ is that in problems with long solutions, not all states are reachable from the initial state. In the case of the problems above, 48 out of 64 states are reachable. If we do make all states reachable, then solutions are considerably shorter. Intuitively, this is because adding a single STRIPS$^1_1$ action has the effect of connecting many pairs of states, making it hard to add many actions without creating shortcuts.

Following the length 38 solution for $n = 6$, we derived a general

Definition 3 *(STRIPS$^1_1$ Planning Task $\mathcal{Q}_n$) Let $\mathcal{Q}_n$ be the STRIPS$^1_1$ planning task including the following $3n + 1$ actions*

- *for all $i \in \{0, 2, 3 \ldots, n-1\}$ we include STRIPS$^1_1$ action $(pre(i), add(1))$;*
- *for all $i \in \{1, 2, \ldots, n-1\}$ we include STRIPS$^1_1$ action $(npre(i), del(i-1))$;*
- *for all $i \in \{1, 2, \ldots, n-2\}$ we include STRIPS$^1_1$ action $(pre(i), add(i+1))$;*
- *STRIPS$^1_1$ actions $(pre(n-4), add(0))$, $(npre(0), del(n-4))$, and $(npre(1), del(n-3))$*
- *STRIPS$^1_1$ actions $(pre(n-1), del(0))$, $(pre(0), del(n-1))$, and $(npre(n-1), add(n-2))$*

k	3	3	4	4	5	5	6	6	7	7	8
n	6	7	8	9	10	11	12	13	14	15	16
l_P	30	35	49	56	72	81	99	110	130	143	165
l_Q	**38**	**47**	**57**	**65**	74	83	92	101	110	119	128
l_R	31	38	52	59	**77**	**86**	**106**	**117**	**139**	**152**	**176**

Table 3: Growth of optimal solution length l_p, l_q and l_r wrt. n and k in $\mathcal{P}_n$, $\mathcal{Q}_n$ and $\mathcal{R}_n$, respectively.

Definition 4 *(STRIPS$^1_1$ Planning Task $\mathcal{R}_n$) Let $\mathcal{R}_n$ be the STRIPS$^1_1$ planning task including the above actions, and the following actions:*

- $pre(i-1), add(i)$ *(add i whenever $i-1$ is true)*
- $npre(i-1), del(i)$ *(delete i whenever $i-1$ is false)*
- $npre(n-1), add(0)$ *(add 0 when $n-1$ is false)*
- $pre(n-1), del(0)$ *(add 0 when 2 is false)*
- $pre(n-1), add(x)$ *(add y when x is true), with $x = n-3$ for all $n \notin \{7, 9\}$ where $x = n-2$.*

While for small values of n the system $\mathcal{Q}_n$ performs better than $\mathcal{P}_n$ and $\mathcal{R}_n$ Table 3 illustrates that for larger values of n this no longer is the case. While $\mathcal{R}_n$ strictly outperforms $\mathcal{P}_n$ the optimal solution lengths grow only by a linear amount.

```
enumerate(action)
  if (action == 0)
   distance = solve();
   if (distance > radius)
    radius = distance;
   return;
  for (pre = NEG(N)+1; pre< POS(N); pre++)
   for (eff = NEG(N)+1;eff< POS(N); eff++)
    a = A-action;
    if (constraint(a,pre,eff,N)) continue;
     for (b=N;b<a;b++)
      if (precondition[b] == POS(pre) ||
          precondition[b] == NEG(pre))
       continue;
    precondition[a] = pre; effect[a] = eff;
    alreadyseen = false;
    for (j = 0;j<a;j++)
     if (precondition[j] == pre && effect[j] == eff)
      continue;
    solve();
```

Figure 2: Generating actions for the STRIPS$^1_1$ planning tasks with $2N+1$ actions to be solved, enumerated in recursive fashion, pruned with constraints on pre- and postconditions.

Fig. 3 shows an option to implement a complete breadth-first search to find the maximum distance between vertex $(0, \ldots, 0)$ to all other vertices, i.e., the eccentricity (or radius) of the problem, including path solution reconstruction link (edge). Compressing and decompressing hypercube nodes from bitvector to ID number representation is optional, one can manipulate the bits on the numbers directly.

Fig. 4 with rollout function displayed in Fig. 4 and policy adaptation routing in Fig. 4 displays our Nested Rollout Policy Adaptation (NRPA) implementation for the STRIPS$^1_1$

```
solve()
   dist[0] = 0; dmax = 0; table[0] = true; q.enqueue(0);
   while (!q.empty())
    v = q.dequeue(); decompress(v);
    for (a = 0; a < A; a++)
     for (i = 0; i < N;i++)  // add-effect with pos-prec
      if (precondition[a]== POS(i) && v[i]==1)
       for (j = 0; j < N;j++)
        if (effect[a]== POS(j) && v[j] == 0)
         v[j] = 1; w = compress(v);
         if (!table[w])
          edge[w] = v; table[w] = true;
          q.enqueue(w); dmax = dist[w] = dist[v] + 1;
         v[j] = 0;
    for (i = 0; i < N;i++)  // add-effect with neg-prec
     if (precondition[a]== NEG(i) && v[i]==0)
      for (j = 0;j < N;j++)
       if (effect[a] == POS(j) && v[j] == 0)
        v[j] = 1; w = compress(v);
        if (!table[w])
         edge[w] = v; table[w] = true;
         q.enqueue(w); dmax = dist[w] = dist[v] + 1;
        v[j] = 0;
    for (i=0;i < N;i++) // del-effect with pos-prec
     if (precondition[a] == POS(i) && v[i] == 1)
      for (j = 0; j < N;j++)
       if (effect[a]== NEG(j) && v[j] == 1)
        v[j] = 0; w = compress(v);
        if (!table[w])
         edge[w] = v; table[w] = true;
         q.enqueue(w); dmax = dist[w] = dist[v] + 1;
        v[j] = 1;
   for (i = 0; i < N;i++)  // del-effect with neg-prec
    if (precondition[a] == NEG(i) && v[i]==0)
     for (j = 0; j<N;j++)
      if (effect[a]== NEG(j) && v[j] == 1)
        v[j] = 0; w = compress(v);
        if (!table[w])
         edge[w] = v; table[w] = true;
         q.enqueue(w); dmax = dist[w] = dist[v] + 1;
        v[j] = 1;
 return dmax;
```

Figure 3: Optimal STRIPS$^1_1$ planning with BFS.

problem that we used to improve $\mathcal{P}$ to find $\mathcal{R}$. For an introduction to this Monte-Carlo algorithm we refer the reader to (Rosin 2011)). The main search routing is a recursion tree with complete rollout procedure to be called at the leaves to generate the action in the system, which is evaluated by our BFS. The policy which actions is followed by another is adapted and refreshed on each level on better solutions found, which in turn changes the bias for selecting the successors in the rollout.

Conjecture

We have 2^n vertices and $n2^{n-1}$ edges in the hypercube H_n. Every single STRIPS$^1_1$ action fixes two bits: one on the source vector and one in the target vector, and the one in the target vector must flip, so its base, corresponds to a hy-

```
Pair NRPA(level)
  best.score = 0;
  if (level == 0)
    best.score = rollout();
    best.actions = base;
  else
    backup[level] = global;
    for (i=0; i < iterations; i++)
      r = NRPA(level - 1);
      if (r.score > best.score)
        best.score = r.score;
        best.actions = r.actions;
        adapt(best.actions,level);
    global = backup[level];
  return best;
```

Figure 4: STRIPS$_1^1$ planning with NRPA.

```
adapt(selection,level)
  visited = [false..false];
  successors = 0;
  for (pre = NEG(N)+1; pre< POS(N); pre++)
    for (eff = NEG(N)+1;eff< POS(N); eff++)
      if (POS(pre)==POS(eff) || POS(pre)==NEG(eff) ||
          NEG(pre)==POS(eff) || NEG(pre)==NEG(eff))
        continue;
      i = convert(pre,eff);
      if (!visited[i]) moves[successors++] = i;
  for (j=0; j < A-1; j++)
    factor = 1.0;
    node = selection[j];
    backup[level][node][sselection[j+1]] += factor;
    z = 0.0;
    for (i=0; i < successors; i++)
      z += exp(global[node][moves[i]]);
    for (i=0; i < successors; i++)
      backup[level][node][moves[i]] -= factor *
        exp(global[node][moves[i]])/z;
    visited[selection[j+1]] = true;
```

Figure 5: STRIPS$_1^1$ policy adaption in NRPA.

percube H_{n-2} with 2^{n-2} vertices (and $(n-2)2^{(n-2)-1} = (n-2)2^{n-3}$ edges).

The number of the m-dimensional hypercubes (alias m-cubes) contained in the boundary of an n-cube is $2^{n-m}\binom{n}{m}$ with $m = n-2$ we have

$$2^{n-(n-2)}\binom{n}{n-2} = \frac{4(n!)}{2(n-2)!} = 2n(n-1)$$

different cubes H_{n-2} in H_n.

With n variables, positive or negative in preconditions and effects, there are at most $(2n)(2n) = 4n^2 = \Theta(n^2)$ different STRIPS$_1^1$ actions. As we do not allow repeating variables, then this value reduces to $2n(2(n-1)) = 4n^2 - 4n = \Theta(n^2)$. This is exactly twice the number of hypercubes H_{n-2} as preconditions and effects can be exchanged without changing the cube. Otherwise each STRIPS$_1^1$ action selects between two orientations of H_{n-2}, which we denote

```
rollout()
  visited = (false...false);
  baseSize = 0;
  node = base[baseSize++] = init();
  visited[node] = true;
  while (baseSize < A)
    successors = 0;
    for (pre = NEG(N)+1;pre< POS(N);pre++)
      for (eff = NEG(N)+1;eff< POS(N);eff++)
      if (POS(pre)==POS(eff) || POS(pre)==NEG(eff) ||
          NEG(pre)==POS(eff) || NEG(pre)==NEG(eff))
        continue;
      i = convert(pre,eff);
        if (!visited[i]) moves[successors++] = i;
    sum = 0;
    for (i=0; i < successors; i++)
      value[i] = exp(global[node][moves[i]]);
      sum += value[i];
    mrand = rand(0..sum);
    i=0; sum = value[0];
    while(sum<mrand) sum += value[++i];
    node = base[baseSize++] = moves[i];
    visited[node] = true;
  baseSize = 0;
  while(baseSize < A)
    (pre,eff) = code(base[baseSize]);
    precondition[baseSize] = pre;
    effect[baseSize++] = eff;
  return solve();
```

Figure 6: STRIPS$_1^1$ rollout in NRPA.

as H^+_{n-2} and H^-_{n-2}. More precisely for each choice of assignments l_i, l_j to the variables i and j, $1 \le i \ne y \le n$, the projection of the undirected graph $H_n = (V_n, E_n)$ to the directed cubes H^+_{n-2} and H^+_{n-2} with

$$H^+_{n-2}(l_i, l_j) = \{(u,v) \in \{0,\dots,n-1\}^2 \mid u_i = l_i \wedge v_j = l_j\}$$

and

$$H^-_{n-2}(l_i, l_j) = \{(u,v) \in \{0,\dots,n-1\}^2 \mid u_i = l_j \wedge v_j = l_i\}.$$

As the problem is symmetric and the variables are interchangeable, without loss of generality, we can assume the travel of the longest shortest path to start at $(0,\dots,0)$, so that the problem to find the STRIPS$_1^1$ actions generating the longest travel distance in fact equivalent to finding the eccentricity of the graph starting at $(0,\dots,0)$.

Our conjecture implying the assertion that STRIPS$_1^1$ is NP-complete is the following.

Conjecture. *In the hypercube H_n, for any selection S of the $4n(n-1)$ subcubes $H^d_{n-2}(l_i, l_j)$ with given literal assignments l_i and l_j and orientation $d \in \{+,-\}$, the maximal shortest path length for a path starting at $(0,\dots,0)$ of the graph defined by the union of S is polynomial.*

Related Work

Nebel studied the *short solution hypothesis* to prove NP-completeness in the different context of multiagent path

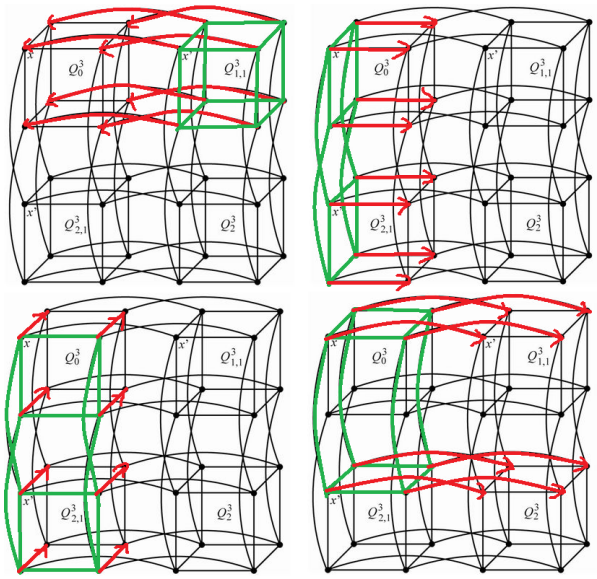

Figure 7: Sketch of four different STRIPS$^1_1$ actions in the 5-dimensional hypercube, green lines indicating the start and arrows indicating the end of the actions, there are $2^{n-2} = 8$ edges associated edges with each action and there are overlaps.

finding on directed graphs (diMAPF) (Nebel 2023). While diMAPF has been shown to be polynomial for some special cases, it has been established that the problem is NP-hard in general. To prove NP-completeness, Nebel showed diMAPF to be in NP by the fect that the short solution hypothesis for strongly connected directed graphs holds.

There is work by Domshlak on recursively constructed directed hypercubes (Domshlak 2002). It is shown by construction that these graphs have one source and one sink and while the diameter is exponential in n, the distance from source to any node is at most n and from any node to the sink, too. Unfortunately, the direction in the hypercube defined by STRIPS$_{1,1}$ actions is not recursively defined, but by directed hypercubes of dimension $n - 2$.

Everett and Gupta showed that there directed hypercubes that are acyclic and that have exponential path length, with a construction that is based on Fibonacci numbers (Everett and Gupta 1989). The main observation is that edges of any (chordless) path can be oriented so that this paths is shortest.

Gregor studied hypercubes in his PhD thesis including the findings of Hamiltonian cycles and the inclusion of faulty edges to be avoided (Gregor 2006). He also looks at Fibonacci cubes as subcubes of the hypercube, where the only labels that are allowed are bitstrings with no two consecutive 1 bits.

Conclusion

Bylander (1994) settled many questions about the computational complexity of STRIPS, with some carrying over to planning with SAS$^+$ (Bäckström and Nebel 1993).

Determining the complexity of STRIPS$^1_1$ was left open and remains a fascinating research topic.

We studied the problem to some depth. At the end, we could not prove the existance or non-existance of exponentially long plans for STRIPS$^1_1$. We found a STRIPS$^1_1$ planning task with n propositions, were shortest plans was of length $\Omega(n^2)$, but also found systems with larger shortest plans. If $\Omega(n^2)$ is the worst possible, the STRIPS$^1_1$ problem is in NP, and following Bylander, NP-complete.

What is the practical value of the result? If STRIPS$^1_1$ is polynomial it may serve as a heuristic estimation function, and if it is NP-complete, similar to the relaxed plan graph heuristic h^+ (Hoffmann and Nebel 2001), we might approximate STRIPS instances with STRIPS$^1_1$ ones to compute a heuristic value.

Acknowledgements

Thanks to Bernhard Nebel highlighting that Anders Jonsson et. al. showed that STRIPS$^1_2$ is PSPACE-complete and that exponential lower bounds on the plan length to proove NP containment have to be dealt with care, as there could be other polynomial certificates. The presented work has been supported by the Czech Science Foundation (GAČR) under the research project number 22-30043S.

References

Abbott, H. L., and Katchalski, M. 1988. On the snake in the box problem. *J. Comb. Theory B* 45(1):13–24.

Abbott, H. L., and Katchalski, M. 1991. Further results on snakes in powers of complete graphs. *Discret. Math.* 91(2):111–120.

Bäckström, C., and Nebel, B. 1993. Complexity results for SAS+ planning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, 1430–1435.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artif. Intell.* 69(1-2):165–204.

Dang, C.; Bazgan, C.; Cazenave, T.; Chopin, M.; and Wuillemin, P. 2023. Warm-starting nested rollout policy adaptation with optimal stopping. In Williams, B.; Chen, Y.; and Neville, J., eds., *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, 12381–12389. AAAI Press.

Domshlak, C. 2002. On recursively directed hypercubes. *Electron. J. Comb.* 9(1).

Edelkamp, S., and Cazenave, T. 2016. Improved diversity in nested rollout policy adaptation. In *KI 2016: Advances in Artificial Intelligence - 39th Annual German Conference on AI, Klagenfurt, Austria, September 26-30, 2016, Proceedings*, 43–55.

Edelkamp, S.; Jabbar, S.; and Lluch-Lafuente, A. 2006. Heuristic search for the analysis of graph transition systems. In *Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, September 17-23, 2006, Proceedings*, 414–429.

Everett, H., and Gupta, A. 1989. Acyclic directed hypercubes may have exponential diameter. *Information Processing Letters* 32(5):243–245.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artif. Intell.* 2(3/4):189–208.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.* 20:61–124.

Gregor, P. 2006. *Subgraphs of Hypercubes – Embeddings with Restrictions or Prescriptions*. Ph.D. Dissertation, Charles University, Faculty of Mathematics and Physics.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res.* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.* 14:253–302.

Jonsson, A.; Jonsson, P.; and Lööw, T. 2014. Limitations of acyclic causal graphs for planning. *Artificial Intelligence* 210:36–55.

Katz, M., and Domshlak, C. 2008. Structural patterns heuristics via fork decomposition. 182–189.

McDermott, D. V. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2):35–55.

Nebel, B. 1999. Compilation schemes: A theoretical tool for assessing the expressive power of planning formalisms. In *KI-99: Advances in Artificial Intelligence, 23rd Annual German Conference on Artificial Intelligence, Bonn, Germany, September 13-15, 1999, Proceedings*, 183–194.

Nebel, B. 2023. The small solution hypothesis for MAPF on strongly connected directed graphs is true. In Koenig, S.; Stern, R.; and Vallati, M., eds., *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, Prague, Czech Republic, July 8-13, 2023*, 304–313. AAAI Press.

Palombo, A.; Stern, R.; Puzis, R.; Felner, A.; Kiesel, S.; and Ruml, W. 2015. Solving the snake in the box problem with heuristic search: First results. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel.*, 96–104.

Rosin, C. D. 2011. Nested rollout policy adaptation for monte carlo tree search. In Walsh, T., ed., *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 649–654. IJCAI/AAAI.