Computing Upper Bounds on Lengths of Transition Sequences

Jussi Rintanen[*]
Department of Information and Computer Science
Aalto University, Finland

Charles Orgill Gretton[†]
Optimisation Research Group
NICTA, Australia

Abstract

We describe an approach to computing upper bounds on the lengths of solutions to reachability problems in transition systems. It is based on a decomposition of state-variable dependency graphs (causal graphs). Our approach is able to find practical upper bounds in a number of planning benchmarks. Computing the bounds is computationally cheap in practice, and in a number of benchmarks our algorithm runs in polynomial time in the number of actions and propositional variables that characterize the problem.

1 Introduction

Lower bounds on lengths of action sequences have been studied extensively in the AI literature [Culberson and Schaeffer, 1996; Korf, 1997; Haslum and Geffner, 2000; Bonet and Geffner, 2001; Dräger *et al.*, 2009], in connection with optimal search algorithms such as $A^\star$ [Hart *et al.*, 1968]. Studies of *upper bounds* are less common and more limited. In this work, by *upper bound* N we mean that, if a solution to the problem exists, the length of an optimal solution is no greater than N. An exponential upper bound for a planning problem described using N Boolean state variables is $2^N - 1$. Tighter bounds are known for restricted cases. For example, if actions only have positive preconditions and effects, no action ever needs to be taken more than once. In that case the number of actions is an upper bound on the shortest plan [Bylander, 1994]. Despite this and other important insights about restricted classes of actions [Brafman and Domshlak, 2003; Giménez and Jonsson, 2012], a procedure for inferring upper bounds for a broad range of problems has remained elusive.

[*]Also affiliated with the Institute for Integrated and Intelligent Systems, Griffith University, Australia, and the Helsinki Institute of Information Technology, Finland. This work was funded by the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, 251170).

[†]Also affiliated with the Artificial Intelligence Group, Australian National University, and the Institute for Integrated and Intelligent Systems, Griffith University, Australia. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

There are several uses for upper bounds in solving reachability problems. First, there is the detection of (sub)problems that have no solution. In explicit state-space search, if for a state s the distance from s to a goal state has a lower bound l and an upper bound u with $u < l$, then the goal state is not reachable from s, and s can be pruned from the search. In the case of the planning as satisfiability approach [Kautz and Selman, 1996], a sequence of satisfiability tests, for example testing the existence of plans of lengths $1, \leq 2, \leq 3$, and so on, is performed. Tight upper bounds yield a practical termination test for this procedure, because once we have tested at the upper bound, if no solution was found, then none exists.

A second application is limiting the search effort in approaches that reduce the overall reachability problem to a sequence of fixed-length reachability problems, as in planning-as-SAT and related approaches. The efficiency of this type of plan search lies in the strategy used to focus search effort on different horizon lengths [Rintanen, 2004; Streeter and Smith, 2007]. These strategies benefit from having tight upper and lower plan length bounds. The Streeter and Smith procedure performs a binary search over horizon lengths, and relies on tight initial bounds. The algorithms by Rintanen do not require tight upper bounds, however they can avoid a great deal of unnecessary search where a tight upper bound is given.

In this paper we have taken the first step towards a general approach to computing such upper bounds. Section 2 formally defines the succinct reachability problem. In Section 3, we recall the simplest and best known exponential upper bounds obtained from the cardinality of the state space, discuss exponential upper bounds obtained from the cardinality of the action set, and discuss how invariants can inform better bounds. In Section 4 we propose a decomposition of the set of state variables according to a dependency graph. We describe an approach which analyzes that graph to compute plan length bounds. In Section 5 we evaluate that procedure with a number of planning benchmarks used in the International Planning Competitions (IPC) from 1998 to 2011. Finally, in Section 6 we conclude the paper by pointing to future research directions.

2 Preliminaries

We formalize the planning problem as a 4-tuple $\Pi = \langle X, I, A, G\rangle$ where X is a finite set of Boolean state variables, which induces a state space $\mathcal{S}(X)$ consisting of all

states $s : X \to \{0,1\}$ that are total functions from the state variables to 0 and 1, $I : X \to \{0,1\}$ is *the initial state*, A is a finite set of actions (p,e) where *the precondition* p and *the effects* e are consistent sets of literals over X, and G, *the goal*, is a set of literals over X.

Given a state s, its successor $s' = \text{exec}_a(s)$ with respect to action $a = (p,e) \in A$ is the state that satisfies $s' \models e$ and $s'(x) = s(x)$ for all $x \in X$ such that x does not occur in e. This is defined iff $s \models p$. An action sequence $a_1, \ldots, a_n$ is executable (in state s) if $\text{exec}_{a_n}(\text{exec}_{a_{n-1}}(\cdots \text{exec}_{a_2}(\text{exec}_{a_1}(s))))$ is defined. A plan for $\Pi = \langle X, I, A, G\rangle$ is a sequence $a_1, \ldots, a_n$ of actions from A such that $\text{exec}_{a_n}(\text{exec}_{a_{n-1}}(\cdots \text{exec}_{a_2}(\text{exec}_{a_1}(I))\cdots)) \models G$.

We can *project* a planning problem $\Pi = \langle X, I, A, G\rangle$ to a subset $X' \subseteq X$ of state variables, obtaining $\Pi \downarrow X' = \langle X', I \downarrow X', \{a \downarrow X' | a \in A\}, G \downarrow X'\rangle$, where $I \downarrow X' = \{(x,v) | (x,v) \in I, x \in X'\}$ by limiting the state to variables in X', $(p,e) \downarrow X' = (p',e')$ obtained from $(p,e) \in A$ by removing all literals with a state variable in $X \backslash X'$ from both p and e. Finally $G \downarrow X'$ is obtained from G by deleting literals with variables not in X'.

Further, we can project an action sequence $a_1, \ldots, a_n$ to a set $X' \subseteq X$ of state variables by projecting the actions to X', obtaining $a_n \downarrow X', \ldots, a_n \downarrow X'$ and then deleting those projected actions that have empty effects (that is, the actions that do not have any effect on variables in X').

Our method for deriving upper bounds is based on a decomposition of a directed graph into its strongly connected components (SCC). An SCC of a directed graph is a subgraph in which there is a directed path from each vertex to every other vertex. A successor of an SCC S is an SCC S' so that, in the underlying graph, there is an arc from a vertex in S to a vertex in S'. A predecessor of S is a component which has S as a successor. The ancestors of S are the SCCs, other than S, for which there is a directed path in the underlying graph from a vertex in the component to one in S. In this work we shall often identify an SCC with the set of vertices in that subgraph.

3 Plan Length Upper Bounds

A plan length upper bound can be obtained from the cardinality of the state space by observing that a shortest path in the transition graph never visits any state twice: if a state is visited twice, all the actions between these two occurrences of the state could be removed to obtain a shorter plan.

Lemma 1 *For a problem over Boolean variables X s.t. $N = |X|$, the shortest plan cannot be longer than $2^N - 1$. That bound is tight: one can give a problem with N actions which increment an N-bit counter by one, depending on the number of least significant bits that are* 1.

Another upper bound can be obtained from the number of actions. It may be useful when $|X|$ is large relative to $|A|$, which is the situation when actions have preconditions and effects that refer to a high number of state variables.

Lemma 2 *If there are $m = |A|$ actions and a plan exists, then the length of the shortest plan is at most $\sum_{i=0}^{m} \frac{m!}{(m-i)!} - 1$. This bound is tight.*

Proof: Any state reached by a (finite or infinite) sequence of actions from A is uniquely determined by the order of the last occurrences of each of the m actions. The truth value of any state variable, after taking a sequence of actions in the initial state, is the last value assigned to it, or if no action has affected it, its initial value. In other words, I and the order of last occurrence of each action determines the reached state uniquely. As there are at most $m!$ relevant action sequences, there can be at most $m!$ different states. When any i of the m actions are used, there are $\frac{m!}{(m-i)!}$ different action sequences, and we of course have to consider all i between 0 and m to account for all states reachable by plans that contain at most m different actions. For tightness, the reader can verify that, in the following example the bound equates to the number of reachable states, and that a plan in this example might be equal in length to the number of reachable states minus one. Take $A = \{a_1, .., a_m\}$ and $X = \{x_1, .., x_m\} \cup \{y_{i,j} | 1 \leq i < j \leq m\}$. Actions are $a_j = (\emptyset, \{x_j, y_{j,j+1}, .., y_{j,m}, \neg y_{1,j}, .., \neg y_{j-1,j}\})$ so that each action j sets $x_j = y_{j,k} = 1 \; \forall k \in \{j+1, ..., m\}$ and $y_{k,j} = 0$ $\forall k \in \{1, ..., j-1\}$. □

Further assumptions about the properties of individual actions or their interrelations are needed to derive tighter upper bounds. We shall not explicitly consider actions with conditional (state-dependent) effects. Bounds based on the properties of state variables remain valid in that case, but those based on action set cardinality do not. With conditional effects it is possible to represent an increment action, which can be repeated $2^N - 1$ times to increase an N-bit counter from 0 to $2^N - 1$. Hence a small number of actions does not necessarily yield practically significant upper bounds.

3.1 Invariants

Invariants [Gerevini and Schubert, 1998; Rintanen, 2008] represent dependencies between state variables in the reachable state space. They characterize unreachable states, and have been used to derive compact state representations for explicit state-space search. Most invariants used in practice can be expressed as clauses with one or two literals (unit clauses l and binary clauses $l \vee l'$). We employ 2-literal invariants to derive tighter plan length bounds.

Lemma 3 *Let $\langle X, I, A, G\rangle$ be a problem instance so that $\neg x \vee \neg x'$ is an invariant for all $\{x, x'\} \subseteq X$ such that $x \neq x'$. A shortest plan has at most n actions, where $n = |X|$.*

Proof: The invariants imply that one state variable can be true at any state reachable from I. The number of reachable states is at most $|X| + 1$ (including the state in which all of the state variables are false). A plan visiting a subset of these states, with no state visited twice, has at most $|X|$ actions.[1] □

[1] In many problems arising in practice there is a guarantee that at least one of the state variables is true, for example because any

The structure of the set of invariants in the above lemma is specific. For arbitrary sets of invariants, each taking the form of a 2-literal clause, looser upper bounds hold. Tractable exact model-counting is possible for sets of 2-literal clauses satisfying certain graph-theoretic properties [Luna *et al.*, 2007], but these properties do not seem to be of practical relevance to 2-literal invariants arising in typical planning problems. In any case, we can employ general model-counting for Boolean 2SAT—i.e. clauses have at most 2 literals— to obtain upper bounds based on the cardinality of the state space by leveraging problem invariants.

4 Bounding via Dependency Graph Analysis

For many problems that occur in practice, the discussed bounds are impractically loose. We now describe how dependencies between problem variables can be expressed using a graph, and leverage problem decompositions suggested by that graph in order to obtain tight bounds. In particular, we shall describe and analyse a type of *dependency graph* that suggests a decomposition of the problem into subproblems. The concept of a dependency (often termed *causal*) graph was first described by [Knoblock, 1994] and [Williams and Nayak, 1997], and has since found myriad uses in both theoretical and practical works. Intuitively, dependency graphs express dependencies between state variables in terms of the ability of one variable to impact the value of another. For some classes of problems such graphs expose small subproblems with few interconnections, with the subproblems' size independent of parameters determining the overall problem size. We shall explore how upper bounds on the lengths of solutions to subproblems yield a bound for the underlying problem.

Definition 1 *The dependency graph of a planning problem is the directed graph $G = (V, E)$ that satisfies the following. The vertices V are the set of state variables. There is an arc $(v, v') \in E$ iff either*

1. *there is an action whose effect is described using a literal over v' (i.e. using either v' or $\neg v'$), and a literal over v is a precondition of that action; or*
2. *literals over v' and v occur together in the description of an action effect, and v occurs both positively and negatively in the effects of problem actions.*

Definition 1 departs from a number of other definitions of dependency graphs. In particular, the *classical* causal graph includes the arc (v, v') if v' and v occur together in the description of an action effect.

We illustrate Definition 1 in Figure 1, which depicts the dependency graph for a simple instance of the *logistics* benchmark from the IPC in 2000. The problem we consider has one package, p, one aeroplane, ap, three trucks, $t1$, $t2$, and $t3$, and three cities. Each city contains one of the airport locations, $a1$, $a2$, and $a3$, and one of the corresponding inner-city locations $c1$, $c2$, and $c3$. The aeroplane can visit every airport location. The fact that ap is visiting $a1$ is expressed by the Boolean variable $At(ap, a1)$ begin true. Each truck can visit the airport and downtown locations of its home city. For example, the truck $t1$ can be visiting either $c1$ or $a1$. The package can be transported between locations with the appropriate vehicles. For example, ap can transport p between any two airports, and trucks can transport p between an airport and its corresponding inner-city location. The edges of the dependency graph in Figure 1 capture the conditions for moving vehicles, and those for loading/unloading the package. For example, to move from location $c1$ to the airport $a1$, the truck $t1$ must be visiting $c1$. For package p to be loaded into a vehicle, to achieve $In(p, t1)$ for example, both the vehicle and p have to be at the same location. Similarly, to unload p to a location, p must of be inside the vehicle we intend to unload, and that vehicle must be visiting the location.

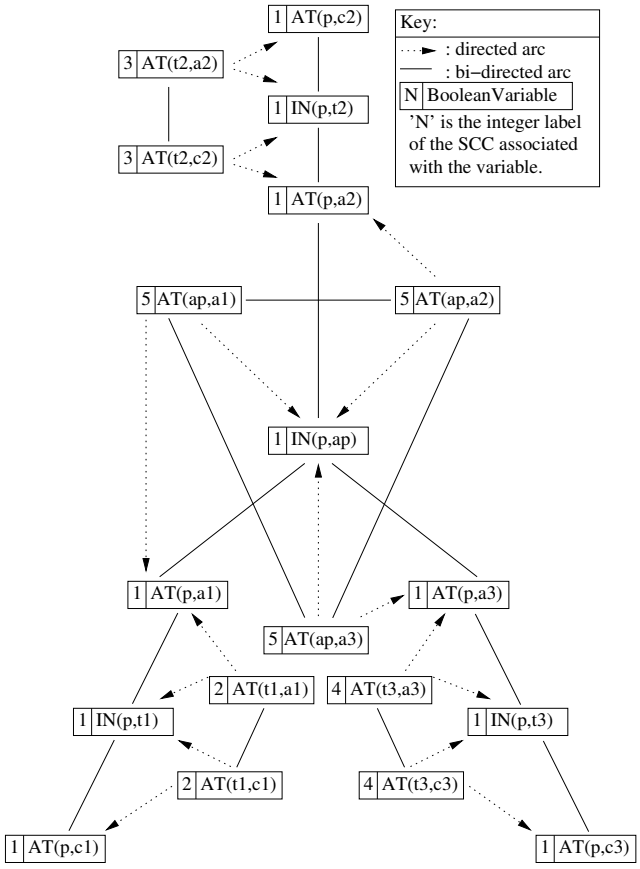

Figure 1: Dependency graph for a *logistics* problem. The graph has five SCC, one component for each of the four vehicles—three trucks and one aeroplane—and one component for the package.

We can decompose a dependency graph into its strongly connected components (SCC) in linear time [Tarjan, 1972].[2] An SCC with no outgoing arcs induces an abstract planning problem with actions that change the state variables in that

action making a state variable false also makes another variable true. Hence the state with all variables in X false is not reachable, and the upper bound on actions is $|X| - 1$ instead of $|X|$.

[2] Assuming that a symbol table for node names with linear-time access has been built, a $O(n \log n)$ computation.

SCC only. Considering SCCs in a planning context more generally, exactly those actions belong to an SCC that change at least one variable in the SCC, and which add/delete only variables in the SCC and its successors.

Continuing with our *logistics* example, in Figure 1 we have labeled each vertex with its corresponding variable, and also with a number that indicates the SCC that vertex participates in. For example, variables *At*(p,ap) and *In*(p,c1) participate in an SCC labeled 1, while *At*(t1,c1) and *At*(t1,a1) participate in 2. Taking $i, j \in \{2..5\}$ and $i \neq j$, the reader can observe that for SCCs labeled i and j respectively, there are no directed paths between vertices in i and j. In other words, any action that alters the location of a vehicle cannot interfere with an action that alters the location of a different vehicle. If the problem goal is to rearrange the vehicles, then we can plan independently for each vehicle. The overall plan for our *logistics* problem in that case need not be longer than 5 steps.

A less contrived example of such independence occurs in the *rovers* benchmark from the IPC in 2006. Here, an autonomous planetary rover must acquire and transmit scientific data which can be acquired by visual analysis (image data), or otherwise by analysis of soil and rock. Acquisition of the former is independent of the latter, and in the SCC decompositions of graphs associated with *rovers* instances, there is one component with actions for soil/rock analysis, and another for image analysis. A plan for such problems can be the composition of a plan for acquisition and transmission of image data followed by a plan for the soil/rock task. In our work we exploit such independence to obtain upper bounds on the lengths of plans in *rovers*, and similar domains.

We must also consider the case where variables in an SCC are dependent on those from a different SCC. We shall proceed with the example in Figure 1 for the moment. Taking a conventional *logistics* goal, of delivering p to a particular location, we are left to address dependencies between the locations of vehicles and our ability to alter the location of p. In order to derive upper bounds for this case, we appeal to an abstraction refinement scheme suggested by the dependency graph, which was previously reported in [Knoblock, 1994]. Observe that there are no directed paths from a vertex in SCC labeled 1 to any vertex with SCC $i \neq 1$. Let S_1 be the set of variables associated with vertices with SCC label 1. If a plan for the logistics problem exists, then a plan shall correspond to a *refinement* of an abstract plan for the problem $\Pi \downarrow S_1$.

Definition 2 *Given a problem* Π*, let* π *be an abstract plan for* $\Pi \downarrow Z$*, and* Z' *a set of problem variables satisfying* $Z \subset Z'$*. A plan* π' *for* $\Pi \downarrow Z'$ *is a* refinement *of* π *iff there exists a* correspondence *function* c *mapping each action in* π *to one in* π'*, satisfying:*

1. $\forall a \in \pi$*, both* $c(a)$ *and* a *are projections of the same action in* Π*.*
2. *If* a_1 *precedes* a_2 *in* π*, then* $c(a_1)$ *precedes* $c(a_2)$ *in* π'*.*
3. $\forall a_3 \in \pi'$ *where* $\nexists a_1 \in \pi$ *s.t.* $c(a_1) = a_3$*, we have* $\exists a_2 \in \pi$*,* $(p, e) = c(a_2)$*,* a_3 *precedes* $c(a_2)$ *in* π'*, there is a precondition literal* $x \in p$ *whose subject is a variable from* $Z' \backslash Z$*, and in every deordering of the serial* π' *into a valid* partially ordered plan*, there is a* causal chain *from* a_3 *to the precondition* x*.*

Generally, where Z are the variables/vertices in an SCC, a plan for the abstract problem $\Pi \downarrow Z$ can be, by a process of iterative refinement, extended into progressively less abstract plans. A bound on the length of the abstract plan for $\Pi \downarrow Z$ can be obtained using the insights of Section 3.1. Each step of iterative refinement adds actions from, *and only from*, the predecessor SCCs. Repeating the bounding exercise that we did for $\Pi \downarrow Z$, we can calculate a bound on the length of such a refinement, and by induction a bound on the length of the plan refined to the point that it is part of a solution to Π.

In later sections we shall develop these ideas formally and algorithmically. For now we further illustrate the above scheme, to show more concretely how counting according to the variable dependencies of a problem can give significantly tighter bounds compared to those of Section 3.1. A package can only be at one place at a time, therefore by Lemma 3 an abstract plan for S_1 in our *logistics* example has upper bound 9.[3] An abstract plan will be of the form $a_1, a_2, .., a_9$ where each a_i is a load/unload action. We must now consider inserting additional transportation actions into that abstract plan. A refinement will have the form $\sigma_1, a_1, \sigma_2, a_2, .., \sigma_9, a_9$, where σ_i terms are sequences of actions from SCCs with indices in 2..5, for example drive and fly actions. We have already seen that each σ_i is of length at most 5, therefore a plan length upper bound for our *logistics* example is 54.

4.1 Formalization

Where Π is a problem, we write $\ell(\Pi)$ for the length of the longest optimal execution in Π, that is, the maximum possible length of a shortest possible action sequence in Π from the initial state to another state in Π. Notice that $\ell(\Pi)$ is independent of the goals. Writing $\pi(s)$ for the set of all executions in Π from the initial state I to s and $|\pi|$ for the length of the plan π,

$$\ell(\Pi) = \max_{s \in \mathcal{S}(X)} \min_{\pi \in \pi(s)} |\pi|.$$

Lemmas 1, 2 and 3 provide bounds on $\ell(\Pi)$. For the longest plan, written $\ell(\Pi; G)$, we have

$$\ell(\Pi; G) = \max_{s \models G} \min_{\pi \in \pi(s)} |\pi|.$$

Note that $\ell(\Pi; G) \leq \ell(\Pi)$. Given a strongly connect component S, we write $\mathcal{P}(S)$ for the set of all vertices in the ancestors of S. Where S is an SCC with vertices Y, we write $\Pi \downarrow S$ to denote the underlying problem projected to Y.

Theorem 4 *Let* $\mathfrak{S}$ *be the set of components in the dependency graph for* Π *which have no successors. Then we have*

$$\ell(\Pi) \leq \Sigma_{S \in \mathfrak{S}} \ell(\Pi \downarrow S \cup \mathcal{P}(S)).$$

Moreover, for each $S \in \mathfrak{S}$ *we have*

$$\ell(\Pi \downarrow S \cup \mathcal{P}(S)) \leq (\ell(\Pi \downarrow S) + 1)\ell(\Pi \downarrow \mathcal{P}(S)) + \ell(\Pi \downarrow S).$$

[3] A tighter bound of 6 might be observed, however that requires inference beyond the simple counting of Section 3.1.

Proof: A sketch of our proof follows. First, consider the situation where the dependency graph corresponds to a classical causal graph. In that case there is always an edge (v, v') whenever literals over v and v' occur in the same action effect. Every refinement of a plan $\pi \downarrow Z$ is an *ordered refinement* (see [Knoblock, 1994, Definition 4]) provided there is no directed path in that graph from a variable in Z to any variables added by that refinement. In this case our result follows from the fact that layered plan refinement prescribed by [Knoblock, 1994] never adds or deletes variables that occur in the abstract plan being refined. We are left to examine the following situation. Take an abstract plan $\pi \downarrow Z$, and a refinement $\pi \downarrow Z \cup Y$ so there is no directed path in the dependency graph from a variable in Z to any variable in Y. For some $v \in Z, v' \in Y$ we suppose: (a) both those variables occur together in one or more action effects, and (b) v occurs uniformly positive (negative) in action effects. By definition, the actions added by a legal refinement $\pi \downarrow Z \cup Y$ of $\pi \downarrow Z$ cannot have a precondition that mentions v – otherwise there would be a precondition-effect edge from v to v' violating our stated path condition. In the refinement problem the variable v cannot be a goal (because $v \in Z$) or precondition. Therefore, if there is a refinement $\pi \downarrow Z \cup Y$ to $\pi \downarrow Z$, it must be equal to a refinement $\pi \downarrow Z \cup Y \backslash v$ to $\pi \downarrow Z \backslash v$ in terms of the actions and their ordering. Our result follows, because in bounding the number of actions added during a refinement from $\pi \downarrow Z$ to $\pi \downarrow Z \cup Y$, v need not be considered. □

Theorem 4 informs how we can obtain sub-exponential upper bounds in practice. The term $\ell(\Pi \downarrow S)$ can be approximated using the insights presented in Section 3.1. We can approximate $\ell(\Pi \downarrow \mathcal{P}(S))$ terms by examining the subgraph defined for vertices $\mathcal{P}(S)$. In particular, we can recursively use the above bounding expressions for the smaller problem $\Pi \downarrow \mathcal{P}(S)$, and so on. The recursion terminates when the subgraph over $\mathcal{P}(S)$ has only one SCC S', in which case we approximate $\ell(\Pi \downarrow \mathcal{P}(S))$ again following Section 3.1.[4]

We have *excluded* some inter-effect arcs used in earlier definitions of causal graphs. This poses the question: Can we exclude all arcs that are *purely inter-effect*, in the sense that they are solely included in the graph due to Condition 2 of Definition 1, and still have Theorem 4? The following example shows that at least some of those arcs are required.

Example 1 Consider the actions below, described using the variables $x_0, .., x_{n-1}, \gamma$ and $y_0, .., y_{n-1}, \psi, \psi'$.

For $k \in \{1, .., n-1\}$
$(\{\neg\psi', \neg y_k, y_{k-1}, .., y_0\}, \{y_k, \neg y_{k-1}, .., \neg y_0\})$,
$(\{\neg\psi, \neg y_k, y_{k-1}, .., y_0\}, \{y_k, \neg y_{k-1}, .., \neg y_0\})$, and
$(\{\neg\gamma, \neg x_k, x_{k-1}, .., x_0\}, \{x_k, \neg x_{k-1}, .., \neg x_0, \gamma\})$.
$(\emptyset, \{\neg\gamma, \psi\})$, $(\emptyset, \{x_0, \psi'\})$,
$(\{\neg\psi, y_n, .., y_0\}, \{\neg y_n, .., \neg y_0, \neg\psi'\})$, and
$(\{\neg\psi', y_n, .., y_0\}, \{\neg y_n, .., \neg y_0, \neg\psi\})$.

The actions defined over $x_1, .., x_{n-1}$ count (in binary) from 0 to $2^n - 1$. Setting the least significant bit of the counter to 0 has the effect $\gamma = 1$, requiring (if the goal is to be achieved) setting ψ to true with the action $(\emptyset, \{\neg\gamma, \psi\})$ before further such increments are possible. Setting the least significant bit of the counter to 1 sets the variable ψ' true. Suppose $\neg\psi, \neg\psi' \in G$, and notice that one of those literals must be false so that the plan can affect any of $y_0, .., y_{n-1}$. No plan prefix can include a state s s.t. $s \models \psi \wedge \psi'$. Whenever $\psi = 1$ (resp. $\psi' = 1$), it can only be made false by counting from 0 to $2^n - 1$ using the bits $y_0, .., y_{n-1}$. In other words, each increment of 1 in $x_0, .., x_{n-1}$ is followed by $2^n - 1$ increments in $y_0, .., y_{n-1}$. Hence every change in the first counter forces counting through all values of the second counter, inducing a shortest plan of length $2^n - 1 + (2^n - 1)2^n = 2^{2n} - 1$ for the problem with all variables false in the initial state and all variables except ψ and ψ' true in the goal state. Without pure inter-effect arcs we would get two unconnected SCCs and a too low upper bound, which is the sum of the upper bounds for the two SCCs, and not their product as it should be. ■

[4] If the ancestors of S do not mention goal propositions we have $\ell(\Pi \downarrow S \cup \mathcal{P}(S); G) \leq \ell(\Pi \downarrow S)\ell(\Pi \downarrow \mathcal{P}(S)) + \ell(\Pi \downarrow S)$.

4.2 Algorithm

Our bounding procedure can be outlined as follows.

1. Let a vertex v be the subject of a goal literal $l \in G$ iff $l = v$ or $l = \neg v$. If v is not the subject of a goal literal, and if there is no directed path from v to a vertex which is the subject of a goal literal, then we remove v from the dependency graph. It is straightforward to see that any minimal length plan in the underlying problem Π is a minimal length plan in $\Pi \downarrow X \backslash v$ and *vice versa*.
2. We associate with each SCC S a positive integer value v_S. Intuitively, v_S is equal the maximum length of a shortest (abstract) execution which only changes the values of state variables in S. For v_S we take the minimum bound provided by Lemmas 1, 2 and 3 where applicable, and in situations where some (not all) of the variables in S are mutex, we use model-counting to derive a bound based on the cardinality of that abstract state space.
3. Topologically sort the SCCs and go through them so that the calculations related to an SCC S (summarized as values v_S and below t_S) is carried out after the calculations for all its ancestors have been completed.
4. We associate with each SCC S a positive integer value t_S . Intuitively, t_S is an upper bound on the length of an (abstract) action sequence that affects variables in S and its ancestors, with the purpose of achieving a given valuation (which may be determined by the top-level goal G, or by preconditions in the top-level plan with occurrences of variables in S).

   An upper bound is obtained from v_S by assuming that a maximal number of actions are needed between any two actions in $\pi \downarrow S$. That is, before and after every action in $\pi \downarrow S$, we may need $t_{S'}$ actions from every predecessor SCC S' of S. Hence,

$$t_S = v_S + (1 + v_S) \sum_{S' \prec S} t_{S'}$$

   Above, we multiply the sum by v_S (not $1 + v_S$) if no ancestor of S includes a goal variable. Note that actions

affecting both S and its ancestor S' are counted multiple times.

5. The final step sums the t_S terms for all SCCs S that have no successors. Let $S_1, .., S_m$ be all such SCCs, then a plan length upper bound is $t_{S_1} + .. + t_{S_m}$.

5 Experimental Evaluation

We have implemented our algorithm as a program that takes a planning problem as input, constructs the dependency graph, and then derives an upper bound on the length of a plan. All experiments were performed using a compute cluster of 2 GHz Intel Xeon E6540 nodes. In cases where good bounds according to our decomposition are only available by model-counting, we have used CACHET [Sang *et al.*, 2005] to bound the length of plans for the abstract problems induced by SCCs. Deriving the upper bounds is computationally cheap, whether we use Lemmas 1, 2 or 3, or in situations where we resort to model-counting. In the case of all reported results, the only costly calculation associated with our approach is the detection of 2-literal invariants (taking some dozens of seconds for the largest instances), which is required by Lemma 3. Because that computation is standard in all competitive planners, it is not an additional overhead.

As test material, we used the benchmark sets from the planning competitions. Plan lengths typically vary from some tens to some hundreds of steps. Humans can typically derive relatively tight upper bounds on plan lengths in these domains. The results we obtained with automated bounding are reported in Table 1. We have presented data for the 8 domains where an upper bound can be $\leq$ 2.5k, a horizon at which the state-of-the-art SAT-based planner MP [Rintanen, 2012] can routinely solve problems in under 30 minutes. That $\leq$ 2.5k condition is met for 69 instances, 18 of which are from *logistics*, and another 30 from the trivial domain *movie*. In Figure 5 we explore the relationship between the upper bounds we are able to derive and the number of problem variables. The figure highlights our key claim: Our decomposition can yield sub-exponential growth in upper bounds as the number of variables increases, into the hundreds, and even into the thousands. This is particularly evident in domains *satellite* and *logistics*, where *no* model-counting is necessary, [5] and in *zeno*.

It is worth understanding why our approach works so effectively in *logistics*. There are no capacity constraints on the transports. The location of a package does not alter the possibility of moving transports, and there is no dependence between any two transport's actions either. Also, package movements are mutually independent. A consequence of all this independence is that the number of SCCs in the dependency graph is equal to the number of transports and packages. The number of variables in the SCC associated with a package is equal to the number of places the package can be. Moreover, Lemma 3 applies to every SCC. Each package and each transport can only be in one place at a time.

The benefit of omitting some of the inter-effect arcs in the dependency graph in Definition 1, though broadly important, can be more concretely understood by examining *rovers*. The graph associated with the second problem in that benchmark has 7 SCC if all inter-effect arcs are included. A graph excluding those arcs has 10 SCC, with additional components of size 1 for each of the propositions *have_rock_analysis*, *have_soil_analysis* and *have_image*, that is, all facts that can only become true once, and then remain true. Bounding with all inter-effect arcs yields 235 steps, and 122 steps according to Definition 1. The difference occurs because using the former we are bounding abstract plans that achieve facts, such as *have_image*, multiple times. These facts cannot be made false and hence need only be achieved once, a constraint that cannot be expressed using invariants.

Domain	year	$\leq$		Upper Bound Statistics			
				minimum		maximum	
		2.5k	1M	#SCC	|SCC|	#SCC	|SCC|
*logistics**	00	18	38	48		1728	
				7	7	28	23
*logistics**	98	0	30	NA		NA	
				NA	NA	NA	NA
*movie**	98	30		8		8	
				8	1	8	1
psr	04	3	20	1023		2159	
				4	10	5	14
rovers	06	3	11	122		1562	
				10	4	14	8
*satellite**	04	2	20	171		1285	
				6	7	9	8
storage	06	1	4	255		255	
				5	10	5	10
tpp	06	4	10	129		516	
				4	6	27	16
zeno	02	7	20	254		1990	
				3	10	7	11

Table 1: Columns 1, 2 and 3 give the domain, the year it first featured at an IPC, and the number of instances where our procedure yields a bound $\leq 2.5k$ in the left cell and $\leq 1M$ in the right. We annotate the domain name with '*' if all the bounds are calculated without model-counting. The cells in Column 4 list: (*above*) the shortest bound computed for a domain instance, (*below, left*) the number of SCC in the graph associated with that instance, and (*below, right*) the size of the biggest SCC in the graph for that instance. Column 5 follows the format of Column 4, reporting data regarding the longest bound found that was $\leq 2.5k$.

We can leverage upper bounds in SAT-based planning to prove that a problem admits no solution. We experimented using a version of *rovers* with *qualitative preferences* (QP) from the IPC in 2006. Haslum (2007) proposed an optimal QP solution procedure that operates by solving *open sets*, which are sets of classical goal-oriented STRIPS problems. Whether or not these problems admit solutions allows us to infer plan-cost bounds about the underlying QP problem. We obtained "open sets"—each of which is a set of STRIPS problems—for each of the unsolved QP rovers problems from the competition. Specifically, QP *rovers* problems 06 to 19. Each of the sets contains at least one STRIPS problem for which it was not known whether it admits solutions. In total, we have a corpus of 1440 STRIPS problems.Of those, Mp cannot easily find a plan in 86 problems.

[5] i.e. our approach is tractable in both those domains.

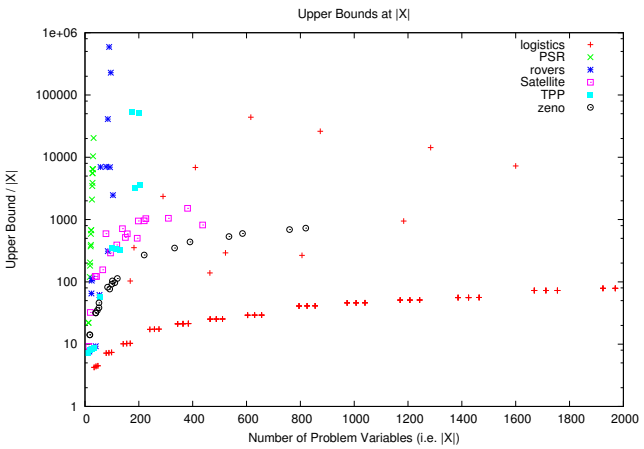

Figure 2: For problems in *logistics*, *psr*, *rovers*, *satellite*, *tpp*, and *zeno* we examine the upper bounds as the number of problem variables increases. Here, *logistics* instances are from the 1998 and 2000 IPCs.

Problem 06 in the QP series has remained open since 2006, i.e. it was not known whether a candidate plan was optimal. In this case, the most challenging STRIPS problem which does not admit a solution has a bound of 3486 steps. In conjunction with our upper bounds, given a 1 GB memory limit and 2 hour time limit MP is able to close *rovers* 06. Our approach is to remove all easily solvable instances from the open sets, and for the remaining instances (none admitting solutions), in each case we prove that no plan exists for a *subset* of the goals. Adding 1 goal literal at a time, in each iteration choosing to add the literal that minimises the upper bound of the resulting instance, MP quickly proves there is no solution. Our approach mitigates upper bound bloat. For example, rather than producing a proof for 3486 steps, which takes days, we produce a proof in hours for 730 steps and 4 (of a possible 11) goal facts. It is worth noting that breadth-first search is ineffective in solving problems from these *open sets*, and that recent admissible landmark-based heuristics offer no guidance here.

Considering planning by forward search of the state space more generally, *admissible* pattern database (PDB) heuristics can be leveraged to prove that no plan exists. First, it is worth noting that given a 4 GB memory limit the state-of-the-art PDB approach by Haslum *et al.* (2007) cannot solve the challenging instances that admit no solutions, including those associated with *rovers* QP problem 06.[6] The allocated memory is quickly exhausted during search. We find that PDB approach can benefit from the iterative goal inclusion procedure outlined above. We prove that 34 of the 86 problems have no solution. Of those 34 problems, 4 are very hard, and cannot be solved using the PDB approach alone due to memory (4 GB) being exhausted. Addressing computation time in this setting, we can report a very significant savings also. Using iterative goal selection, in the 30 STRIPS problems with no plans which can be solved by both our approach and a generic PDB search, we save a total of 646 CPU-seconds. Putting that in perspective, refutation guided by our upper bounds in those problems takes less than 5% of the time taken by a vanilla PDB search.

[6]We used the PDB implementation by Sievers *et al.* (2012).

6 Conclusions and Related Work

Treating a classical (deterministic perfect information) planning setting, we developed an approach to computing upper bounds on the lengths of solutions in transition system problems. This is achieved by first constructing a dependency graph, which expresses dependencies between state variables, and then decomposing that graph into SCCs. A number of earlier works have leveraged dependency graph structures with the motivation of delimiting the boundary between tractable and intractable planning with single-effect actions [Williams and Nayak, 1997; Brafman and Domshlak, 2003; Jonsson and Bäckström, 1998]; a motivation that is orthogonal to ours, of computing upper bounds. In our work, variable independence exhibited by the graphical decomposition is used to obtain an additive expression for an upper bound on the solution length. Factors in that expression are upper bounds on the lengths of solutions to abstract subproblems, obtained in this work according to state space cardinality arguments. We considered two techniques for bounding in subproblems. The first is a polynomial time procedure that appeals to problem invariants and a small fixed set of bounding expressions. The second poses some cardinality questions as 2SAT model-counting problems. Model-counting here, although not tractable, is very efficient. In our experience so far, we have not found that 2SAT model-counting is a bottle neck. In a number of domains where problem variables can be partitioned into components representing physical or virtual objects, the decomposition to SCCs yields several small SCCs, often with a size that is independent of the number of objects and size of the underlying problem. For these types of problems our decomposition method can yield polynomial upper bounds of practical importance. For example, we were able to close an open plan existence problem.

It is also worth noting a previous study related to our own, which developed a different upper bound concept for cost-optimal planning using constraint satisfaction techniques [Cooper *et al.*, 2011]. Those authors addressed the situation where a (suboptimal) plan is already known, and describe how one can then tractably answer the question: Assuming a better (lower cost) plan exists, what is an upper bound on the length of such a plan? Unlike our approach, Cooper *et al.* assume the question of plan existence is *a priori* answered positively. An interesting direction for future work is to use dependency graph decompositions to obtain tighter bounding expressions in that setting.

Wrapping up, there are a few considerations for future research worth highlighting. Numerous planning benchmarks admit well-known tight upper bounds, and yet our procedure is unable to derive them. For example, IPC benchmarks *blocksworld* and *gripper* admit well known linear bounds. An interesting avenue of research is to explore in what situations dependency graph derived bounds can offer tight approximations of the lengths of optimal plans. Also, an im-

portant direction for future research is to derive additional problem constraints with the properties: (a) a solution exists iff a solution to the over-constrained problem exists, and (b) the over-constrained problem yields tight bounds (e.g. via 2SAT model counting with strong invariant conditions in the over-constrained problem). Constraints on the order in which objects/resources can be used (e.g. *lex-leader*) will likely be fruitful here. Future extensions of our framework should also study upper bounds for parallel plans.

References

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artif. Intell.*, 129(1-2):5–33, 2001.

[Brafman and Domshlak, 2003] R. I. Brafman and C. Domshlak. Structure and complexity in planning with unary operators. *J. Artif. Intell. Res. (JAIR)*, 18:315–349, 2003.

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

[Cooper *et al.*, 2011] M. C. Cooper, M. de Roquemaurel, and P. Régnier. A weighted CSP approach to cost-optimal planning. *AI Commun.*, 24(1):1–9, 2011.

[Culberson and Schaeffer, 1996] J. C. Culberson and J. Schaeffer. Searching with pattern databases. In *Canadian Conf. on AI*, pages 402–416. Springer-Verlag, 1996.

[Dräger *et al.*, 2009] K. Dräger, B. Finkbeiner, and A. Podelski. Directed model checking with distance-preserving abstractions. *STTT*, 11(1):27–37, 2009.

[Gerevini and Schubert, 1998] A. Gerevini and L. K. Schubert. Inferring state constraints for domain-independent planning. In *Proc. 15th National Conf. on Artificial Intelligence*, pages 905–912. AAAI Press, 1998.

[Giménez and Jonsson, 2012] O. Giménez and A. Jonsson. The influence of k-dependence on the complexity of planning. *Artif. Intell.*, 177-179:25–45, 2012.

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

[Haslum and Geffner, 2000] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proc. Int. Conf. on AI Planning Systems*, pages 140–149, 2000.

[Haslum *et al.*, 2007] P. Haslum, A. Botea, M. Helmert, B. Bonet, and S. Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc 22nd AAAI Conf. on Artificial Intelligence*, pages 1007–1012. AAAI Press, 2007.

[Haslum, 2007] P. Haslum. Quality of solutions to IPC5 benchmark problems: Preliminary results. In *ICAPS workshop on the International Planning Competition: Past, Present and Future*, 2007.

[Jonsson and Bäckström, 1998] P. Jonsson and C. Bäckström. Tractable plan existence does not imply tractable plan generation. *Ann. Math. Artif. Intell.*, 22(3-4):281–296, 1998.

[Kautz and Selman, 1996] H. A. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proc. 13th National Conf. on Artificial Intelligence*, pages 1194–1201. AAAI Press, 1996.

[Knoblock, 1994] C. A. Knoblock. Automatically generating abstractions for planning. *Artif. Intell.*, 68(2):243–302, 1994.

[Korf, 1997] R. E. Korf. Finding optimal solutions to Rubik's cube using pattern databases. In *Proc. 14th National Conf. on Artificial Intelligence*, pages 700–705. AAAI Press, 1997.

[Luna *et al.*, 2007] G. Luna, P. Bello López, and M. C. González. New polynomial classes for #2SAT established via graph-topological structure. *Engineering Letters*, 15(2):250–258, 2007.

[Rintanen, 2004] J. Rintanen. Evaluation strategies for planning as satisfiability. In *Proc. 16th European Conf. on Artificial Intelligence*, pages 682–687. IOS Press, 2004.

[Rintanen, 2008] J. Rintanen. Regression for classical and nondeterministic planning. In *Proc. 18th European Conf. on AI*, pages 568–571. IOS Press, 2008.

[Rintanen, 2012] J. Rintanen. Planning as satisfiability: Heuristics. *Artif. Intell.*, 193:45–86, 2012.

[Sang *et al.*, 2005] T. Sang, P. Beame, and H. A. Kautz. Heuristics for fast exact model counting. In *Proc. 8th Intnl. Conf. on Theory and Applications of Satisfiability Testing*, pages 226–240. Springer-Verlag, 2005.

[Sievers *et al.*, 2012] S. Sievers, M. Ortlieb, and M. Helmert. Efficient implementation of pattern database heuristics for classical planning. In *Proc. 5th Annual Symposium on Combinatorial Search*, 2012.

[Streeter and Smith, 2007] M. J. Streeter and S. F. Smith. Using decision procedures efficiently for optimization. In *Proc. 17th Intnl. Conference on Automated Planning and Scheduling*, pages 312–319. AAAI Press, 2007.

[Tarjan, 1972] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[Williams and Nayak, 1997] Brian C. Williams and P. Pandurang Nayak. A reactive planner for a model-based executive. In *Proc. 15th Intnl. Joint Conference on Artificial Intelligence*, pages 1178–1185. Morgan Kaufmann Publishers, 1997.