Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)

A State-Space Acyclicity Property for Exponentially Tighter Plan Length Bounds

Mohammad Abdulaziz,[1,2] **Charles Gretton,**[1,3,4] **Michael Norrish**[1,2]
[1] Australian National University, Canberra, Australia
[2] Data61 Canberra Research Lab., Canberra, Australia
[3] HIVERY, Sydney, Australia
[4] Griffith University, Queensland, Australia

Abstract

We investigate compositional bounding of transition system diameters, with application in bounding the lengths of plans. We establish usefully-tight bounds by exploiting acyclicity in state-spaces. We provide mechanised proofs in HOL4 of the validity of our approach. Evaluating our bounds in a range of benchmarks, we demonstrate exponentially tighter upper bounds compared to existing methods. Treating both solvable and unsolvable benchmark problems, we also demonstrate the utility of our bounds in boosting planner performance. We enhance an existing planning procedure to use our bounds, and demonstrate significant coverage improvements, both compared to the base planner, and also in comparisons with state-of-the-art systems.

Introduction

Core AI planning tasks are to: find a plan which achieves the goal in a transition system, or otherwise prove that none exists. The latter also corresponds to the problem of model-checking safety properties: proving that unsafe states are unreachable. Solution methods for these tasks benefit from knowledge of (sub-)system *upper bounds* on the lengths of possible plans. If N is such a bound, and if a plan exists achieving the goal—or violating the safety property—then that plan need not comprise more than N actions.

Biere et al. (1999) identify the system *diameter* as a conceptually appealing upper bound. The diameter is the longest shortest path between any two states. Approximate and exact algorithms have been developed to calculate the diameter given an explicit (e.g. tabular) representation of the system. Exact algorithms have worse than quadratic runtimes in the number of states (Fredman 1976; Alon, Galil, and Margalit 1997; Chan 2010; Yuster 2010), and approximation approaches have super-linear runtimes (Aingworth et al. 1999; Roditty and Vassilevska Williams 2013; Chechik et al. 2014; Abboud, Williams, and Wang 2016). Such explicit calculation of diameters is prohibitively expensive in the settings of planning and model-checking, where systems are described using factored representations, because the systems' explicit representation is usually exponential in the size of the corresponding factored problem description.

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Practical approaches to calculate bounds for problems described using factored representations are *compositional*. Baumgartner, Kuehlmann, and Abraham (2002) and Rintanen and Gretton (2013) developed procedures to compositionally over-approximate the diameter. Abdulaziz, Gretton, and Norrish (2015) provide a tighter procedure by optimising the order of compositional operations. In detail, where problem descriptions exhibit certain exploitable structures, compositional approaches provide useful approximations of diameter using little computational effort. The concrete system bound is over-approximated, by composing together bounds for abstract subsystems which are calculated with relative ease. The subsystems are *projections* of the concrete system, identified according to acyclic structures in the causal/dependency graph (Williams and Nayak 1997; Knoblock 1994). For example, consider the hotel key protocol from (Jackson 2006, p. 185), which provides a domain that severely challenges state-of-the-art planning procedures designed to discover when no plan exists. Each room in the hotel is independent, in the sense that the state of any room i, and actions affecting it are independent of all rooms $j \neq i$. Compositional bounds scale linearly with the number of rooms. The sum of bounds for abstract systems modelling individual rooms is a bound for the concrete system.

We develop a compositional bounding procedure that combines exploitation of acyclicity in variable dependency structures described by Abdulaziz *et.al.*, with a novel approach to exploiting acyclic state-spaces. This enables the decomposition of a given system into abstract sub-systems that are much smaller than what is attainable using state-of-the-art algorithms. Compared to existing practical approaches, ours can yield exponentially tighter bounds. A (sub-)system has an acyclic state-space structure if no state can be encountered twice during an execution. Although such acyclicity does not hold for many typical *concrete* transition systems, it does occur sufficiently often in the *projections* encountered in compositional bounding to be of interest. In the acyclic case, the diameter is bounded by the sum of the diameters of a series of value-based abstractions we call *snapshots*. A snapshot is an abstract subsystem in which the values of some state variables are fixed. For example, the hotel protocol is acyclic because each key can only be used once, for one room, and by one guest. The concrete system diameter is bound above by the sum of the diameters of each of the possible subsystems (snapshots) where a particular key is used to access a particular room.

We experimentally show that our bounding approach significantly outperforms—both in the tightness of the bounds obtained and in the quality of decomposition—existing approaches. Using the upper bounds computed by that algorithm as horizons for a SAT based planner, we: (i) Prove the unsolvability of problems that cannot be proven using the state-of-the-art state-space search planners and model-checkers. One notable example is the problem of model-checking the safety of the hotel key protocol. (ii) Significantly improve the coverage of the SAT based planner Madagascar MP by using upper bounds rather than its simple query strategies (Rintanen 2012).

HOL4 Proofs and Availability Theorem 2, Theorem 4 and Proposition 5 are proven in the interactive theorem prover HOL4 (Slind and Norrish 2008). All our code, Hotel Key benchmarks, and HOL4 proof scripts will be made available online in the case of acceptance.

Background and Notations

Compositional bounds are defined on *factored transition systems* that are purely characterised in terms of a set of *actions*. From actions we can define a set of *valid states*, and then approach bounds by considering properties of *executions* of actions on valid states. Whereas conventional expositions in the planning and model-checking literature would also define initial conditions and goal/safety criteria, here we omit those features from discussion. Our novel bounds, existing compositional bounds, and the notion of diameter are independent of those features.

Definition 1 (States and Actions)**.** *A state, x, is a finite map from variables—i.e., state-characterizing propositions—to Booleans, i.e. a set of mappings $v \mapsto b$. We write $\mathcal{D}(x)$ to denote $\{v \mid v \mapsto b \in x\}$, the domain of x. For states x_1 and x_2, the union, $x_1 \uplus x_2$, is defined as $\{v \mapsto b \mid \text{if } (v \in \mathcal{D}(x_1)) \text{ then } b = x_1(v) \text{ else } b = x_2(v)\}$. Note how the state x_1 takes precedence. An action is a pair of finite maps, (p, e), where p represents the* preconditions *and e represents the* effects*. For action $\pi = (p, e)$, $\mathcal{D}(\pi) \equiv \mathcal{D}(p) \cup \mathcal{D}(e)$.*

Definition 2 (Execution)**.** *When an action π $(= (p, e))$ is executed at state x, it produces a successor state $\text{ex}(x, \pi)$, formally defined as $\text{ex}(x, \pi) = \text{if } p \nsubseteq x \text{ then } x \text{ else } e \uplus x$. We lift* ex *to lists of actions $\vec{\pi}$, so $\text{ex}(x, \vec{\pi})$ denotes the state resulting from successively applying each action from $\vec{\pi}$ in turn, starting at x, which translates to a path in the underlying state-space.*

We give examples of states and actions using sets of literals. For example, $\{a, \overline{b}\}$ is a state where state variables a is (maps to) true, and b is false and its domain is $\{a, b\}$. $(\{a, \overline{b}\}, \{c\})$ is an action that if executed in a state that has a and $\overline{b}$, it sets c to true. $\mathcal{D}((\{a, \overline{b}\}, \{c\})) = \{a, b, c\}$.

Definition 3 (Factored Transition System)**.** *A set of actions δ constitutes a factored transition system.We write $\mathcal{D}(\delta)$ for the domain of δ, which is the union of the domains of all the actions it contains. Where $\text{set}(\vec{\pi})$ is the set of elements from $\vec{\pi}$, the set of valid action sequences, δ^*, is $\{\vec{\pi} \mid \text{set}(\vec{\pi}) \subseteq \delta\}$. The set of valid states, $\mathbb{U}(\delta)$, is $\{x \mid \mathcal{D}(x) = \mathcal{D}(\delta)\}$. For states x and x', $x \rightsquigarrow x'$ denotes that there is a $\vec{\pi} \in \delta^*$ such that $\text{ex}(x, \vec{\pi}) = x'$.*

Definition 4 (Diameter of δ)**.** *The diameter, written $d(\delta)$, is the length of the longest shortest execution, formally:*

$$d(\delta) = \max_{x \in \mathbb{U}(\delta), \vec{\pi} \in \delta^*} \; \min_{\text{ex}(x, \vec{\pi}) = \text{ex}(x, \vec{\pi}'), \vec{\pi}' \in \delta^*} |\vec{\pi}'|$$

If there is a valid action sequence between any two states, then there is a valid action sequence between them that is no longer than the diameter.

Hotel Key Protocol

We now consider the hotel key protocol from (Jackson 2006). Reasoning about safe and unsafe versions of this protocol is challenging for state-of-the-art AI planners and model-checkers. For example, (Blanchette and Nipkow 2010) prove a version of the protocol unsafe for an instance with 1 room, 2 guests and 4 keys. The problem becomes more challenging for the safe version of the protocol, where the only feasible approach is using interactive theorem provers, as in (Nipkow 2006).

We describe the factored transition system corresponding to that protocol. The system models a hotel with R rooms, G guests, and K keys per room, which guests can use to enter rooms (Figure 1 shows an example with $R = 2$, $G = 2$ and $K = 3$). The state characterising propositions are: (i) $\mathtt{l}_{r,k}$, reception last issued key k for room r, for $0 < r \leq R$ and $(r-1)K < k \leq rK$; (ii) $\mathtt{c}_{r,k}$, room r can be accessed using key k, for $0 < r \leq R$ and $(r-1)k < k \leq rK$; (iii) $\mathtt{g}_{g,k}$, guest g has key k, for $0 < g \leq G$, $0 < k \leq RK$; and (iv) $\mathtt{s}_r$, is an auxiliary variable that means that room r is "safely" delivered to some guest. The protocol actions are as follows: (i) guest g can check-in to room r, receiving key k—$(\{\mathtt{l}_{r,k_1}\}, \{\mathtt{g}_{g,k_2}, \mathtt{l}_{r,k_2}, \overline{\mathtt{l}_{r,k_1}}, \overline{\mathtt{s}_r}\})$; and (ii) where room r was previously entered using key k, guest g can enter room r using key k'—$(\{\mathtt{g}_{g,k'}, \mathtt{l}_{r,k}\}, \{\mathtt{c}_{r,k'}, \overline{\mathtt{c}_{r,k}}, \mathtt{s}_r\})$. Thus, guests can retain keys indefinitely, and there is no direct communication between rooms and reception.

For completeness, we note that this protocol was formulated in the context of checking safety properties. Safety is violated only if a guest enters a room occupied by another guest. Formally, the safety of this protocol is checked by querying if there exists a room r, guest g and keys $k \neq k'$, so that $\mathtt{l}_{r,k'} \wedge \mathtt{c}_{r,k} \wedge \mathtt{g}_{g,k'} \wedge \mathtt{s}_r$. The initial state asserts that guests possess no keys, and the reception issued the first key for each room, and each room opens with its first key. Formally, this is represented by asserting $\mathtt{l}_{r,(r-1)K} \wedge \mathtt{c}_{r,(r-1)K}$ is true for $1 \leq r \leq R, (r-1)K < k \leq rK$, and that all other state variables are false.

We adopt some shorthand notations in order to provide examples of concepts in terms of the hotel key protocol. A variable name is written in upper case to refer to a particular assignment, where the only variable that is true is given by the indices. For example, the assignment $\{\overline{\mathtt{c}_{1,1}}, \mathtt{c}_{1,2}, \overline{\mathtt{c}_{1,3}}\}$—indicating room 1 can be accessed using key 2—is indicated by writing $\mathtt{CK}_{1,2}$. We refer to sets of variables by omitting an index term. For example, $\mathtt{l}_1$ indicates the variables $\{\mathtt{l}_{1,i} \mid 1 \leq i \leq 3\}$.

Abstraction and Dependency

Key abstraction concepts for compositional reasoning are *projection* and *snapshot*.

Definition 5 (Projection)**.** *Projecting an object (a state x, an action π, a sequence of actions $\vec{\pi}$ or a factored representation δ) on a set of variables vs restricts the domain of the object or the components of composite objects to vs. Projection is denoted as $x\lfloor_{vs}$, $\pi\lfloor_{vs}$, $\vec{\pi}\lfloor_{vs}$ and $\delta\lfloor_{vs}$ for a state, action, action sequence and factored representation, respectively. However, for action sequences or transition systems, an action with no effects after projection is dropped* entirely.

Example 1. *Consider the set of variables* ROOM1 $\equiv \mathtt{l}_1 \cup \mathtt{c}_1 \cup \{\mathtt{g}_{1,2}, \mathtt{g}_{1,3}, \mathtt{g}_{2,2}, \mathtt{g}_{2,3}\}$. *The variables* ROOM1 *model system state relevant to the 1st hotel room. Figure 1c shows the projected system $\delta\lfloor_{\text{ROOM1}}$.*

A snapshot models the system when we fix the assignment of a subset of the state variables, removing actions whose preconditions *or* effects contradict that assignment.

Definition 6 (Snapshot)**.** *We write $\|X\|$ to denote the cardinality of the set X. For states x and x', let $\mathsf{agr}(x, x')$ denote $|\mathcal{D}(x) \cap \mathcal{D}(x')| = |x \cap x'|$, i.e. a variable that is in the domains of both x and x' has the same assignment in x and x'. For δ and a state x, the snapshot of δ at x is*

$$\delta\!\downarrow_x \equiv \{(p, e) \mid (p, e) \in \delta \wedge \mathsf{agr}(p, x) \wedge \mathsf{agr}(e, x)\}\lfloor_{\mathcal{D}(\delta)\setminus\mathcal{D}(x)}$$

Example 2. *$\delta\lfloor_{\text{ROOM1}}\!\downarrow_{CK_{1,2}}$ is shown in Figure 1d.*

Acylicity in variable dependency has been exploited in previous research by reasoning about *dependency* (also called *causal*) graph from (Williams and Nayak 1997; Knoblock 1994). We formally describe that graph, reviewing precisely what is meant by dependency in this setting.

Definition 7 (Dependency)**.** *A variable v_2 is dependent on v_1 in δ (written $v_1 \rightarrow v_2$) iff one of the following statements holds:*[1] *(i) v_1 is the same as v_2, (ii) there is $(p, e) \in \delta$ such that $v_1 \in \mathcal{D}(p)$ and $v_2 \in \mathcal{D}(e)$, or (iii) there is a $(p, e) \in \delta$ such that both v_1 and v_2 are in $\mathcal{D}(e)$. A set of variables vs_2 is dependent on vs_1 in δ (written $vs_1 \rightarrow vs_2$) iff all of the following statements hold: (i) vs_1 and vs_2 are disjoint, and (ii) There are $v_1 \in vs_1$ and $v_2 \in vs_2$, where $v_1 \rightarrow v_2$.*

Definition 8 (Dependency Graph)**.** *$\mathcal{G}_{\mathcal{D}(\delta)}$ is a dependency graph of δ, if $\mathcal{D}(\delta)$ are its vertices and $\{(u, v) \mid u \rightarrow v \wedge u, v \in \mathcal{D}(\delta)\}$ are its edges. $\mathcal{G}_{\text{VS}}$ is a lifted dependency graph, if its vertices are some partition P of $\mathcal{D}(\delta)$ and $\{(vs_1, vs_2) \mid vs_1 \rightarrow vs_2 \wedge vs_1, vs_2 \in P\}$ are its edges.*

Example 3. *Figure 1b shows a dependency graph associated with the system from Figure 1a. Let* ROOM2 $\equiv \mathtt{l}_2 \cup \mathtt{c}_2 \cup \{\mathtt{g}_{1,5}, \mathtt{g}_{1,6}, \mathtt{g}_{2,5}, \mathtt{g}_{2,6}\}$. *Figure 1b depicts two connected components induced by the sets* ROOM1 *and* ROOM2, *respectively. One lifted dependency graph would have exactly two unconnected vertices, one being a contraction of the vertices from* ROOM1, *and the other a contraction of those from* ROOM2. *Due to the disconnected structure of the dependency graph, intuitively the sum of bounds for $\delta\lfloor_{\text{ROOM1}}$ and $\delta\lfloor_{\text{ROOM2}}$ can be used to upper bound the diameter of the concrete system.*

[1] Our definition is equivalent to those in (Williams and Nayak 1997; Knoblock 1994; Helmert 2006) in the context of AI planning.

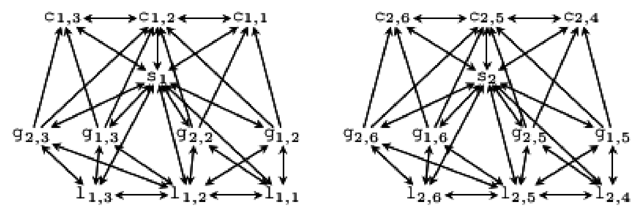
; check in to a room (at reception), receiving a new key

$(\{l_{1,1}\}, \{g_{1,2}, l_{1,2}, \overline{l_{1,1}}, \overline{s_1}\}), (\{l_{1,2}\}, \{g_{1,3}, l_{1,3}, \overline{l_{1,2}}, \overline{s_1}\}),$
$(\{l_{1,1}\}, \{g_{2,2}, l_{1,2}, \overline{l_{1,1}}, \overline{s_1}\}), (\{l_{1,2}\}, \{g_{2,3}, l_{1,3}, \overline{l_{1,2}}, \overline{s_1}\}),$
$(\{l_{2,4}\}, \{g_{1,5}, l_{1,5}, \overline{l_{1,4}}, \overline{s_2}\}), (\{l_{2,5}\}, \{g_{1,6}, l_{1,6}, \overline{l_{1,5}}, \overline{s_2}\}),$
$(\{l_{2,4}\}, \{g_{2,5}, l_{1,5}, \overline{l_{1,4}}, \overline{s_2}\}), (\{l_{2,5}\}, \{g_{2,6}, l_{1,6}, \overline{l_{1,5}}, \overline{s_2}\})$

; enter a room with new key

$(\{g_{1,2}\}, \{c_{1,2}, \overline{c_{1,1}}, s_1\}), (\{g_{2,2}\}, \{c_{1,2}, \overline{c_{1,1}}, s_1\}),$
$(\{g_{1,3}\}, \{c_{1,3}, \overline{c_{1,2}}, s_1\}), (\{g_{2,3}\}, \{c_{1,3}, \overline{c_{1,2}}, s_1\}),$
$(\{g_{1,5}\}, \{c_{2,5}, \overline{c_{2,4}}, s_2\}), (\{g_{2,5}\}, \{c_{2,5}, \overline{c_{2,4}}, s_2\}),$
$(\{g_{1,6}\}, \{c_{2,6}, \overline{c_{2,5}}, s_2\}), (\{g_{2,6}\}, \{c_{2,6}, \overline{c_{2,5}}, s_2\})$

(a)



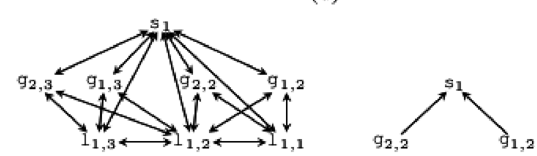
(b)

$(\{l_{1,1}\}, \{g_{1,2}, l_{1,2}, \overline{l_{1,1}}, \overline{s_1}\}), (\{l_{1,2}\}, \{g_{1,3}, l_{1,3}, \overline{l_{1,2}}, \overline{s_1}\}),$
$(\{l_{1,1}\}, \{g_{2,2}, l_{1,2}, \overline{l_{1,1}}, \overline{s_1}\}), (\{l_{1,2}\}, \{g_{2,3}, l_{1,3}, \overline{l_{1,2}}, \overline{s_1}\}),$
$(\{g_{1,2}\}, \{c_{1,2}, \overline{c_{1,1}}, s_1\}), (\{g_{2,2}\}, \{c_{1,2}, \overline{c_{1,1}}, s_1\}),$
$(\{g_{1,3}\}, \{c_{1,3}, \overline{c_{1,2}}, s_1\}), (\{g_{2,3}\}, \{c_{1,3}, \overline{c_{1,2}}, s_1\})$

(c)

$(\{l_{1,1}\}, \{g_{1,2}, l_{1,2}, \overline{l_{1,1}}, \overline{s_1}\}), (\{l_{1,2}\}, \{g_{1,3}, l_{1,3}, \overline{l_{1,2}}, \overline{s_1}\}),$
$(\{l_{1,1}\}, \{g_{2,2}, l_{1,2}, \overline{l_{1,1}}, \overline{s_1}\}), (\{l_{1,2}\}, \{g_{2,3}, l_{1,3}, \overline{l_{1,2}}, \overline{s_1}\}),$
$(\{g_{1,2}\}, \{s_1\}), (\{g_{2,2}\}, \{s_1\})$

(d)



(e) **(f)**

Figure 1: (a) shows the actions of a *transition system* δ representing the hotel key protocol with 2 rooms, 2 guests and 3 keys per room; room 1 is associated with keys 1–3; room 2 with keys 4–6. (b) is the *dependency graph* for that system. (c) is the *projection* of the system on an abstraction that models only the changes related to room 1. (d) is the *snapshot* of $\delta\lfloor_{\text{ROOM1}}$ on $CK_{1,2}$, an abstraction that only analyses the changes related to room 1 when its door recognises key 2 as the current key. (e) and (f) are the dependency graphs of snapshots we have used for illustrative purposes in the main body of the paper. Specifically, we have that (e) is the graph for $\delta\lfloor_{\text{ROOM1}}\!\downarrow_{CK_{1,2}}$, and (f) is the graph for $\delta\lfloor_{\text{ROOM1}}\!\downarrow_{CK_{1,2}}\!\downarrow_{LK_{1,2}}$.

Exploiting Acyclicity in Dependency

Previous authors exploit acyclicity in the dependency graph, a structure that is abundant in practice, for compositional bounding (Baumgartner, Kuehlmann, and Abraham 2002; Baumgartner and Kuehlmann 2004; Rintanen and Gretton 2013; Abdulaziz, Gretton, and Norrish 2015). We review the approach from (Abdulaziz, Gretton, and Norrish 2015), that performs compositional bounding of the *sublist diameter*.

Definition 9 (Sublist Diameter)**.** *Recall that a list $\vec{\pi}'$ is a sublist of $\vec{\pi}$, written $\vec{\pi}' \preceq \vec{\pi}$, iff all the members of $\vec{\pi}'$ occur in the same order in $\vec{\pi}$. The sublist diameter, $\ell(\delta)$, is the length of the longest shortest equivalent sublist to any execution $\vec{\pi} \in \delta^*$ starting at any state $x \in \mathbb{U}(\delta)$. Formally,*

$$\ell(\delta) = \max_{x \in \mathbb{U}(\delta), \vec{\pi} \in \delta^*} \quad \min_{\mathsf{ex}(x,\vec{\pi}) = \mathsf{ex}(x,\vec{\pi}'), \vec{\pi}' \preceq \vec{\pi}} |\vec{\pi}'|.$$

Note, ℓ is an upper bound in the sense $d(\delta) \leq \ell(\delta)$.

Compositional techniques compute upper bounds by composing together bounds for abstract subproblems. To make these ideas concrete, consider the compositional function $\mathsf{N}_{\mathsf{sum}}\langle b\rangle(\delta, \mathcal{G}_{\mathrm{VS}})$, defined via a recurrence below. The functional parameter b is used to bound abstract subproblems, $\mathcal{G}_{\mathrm{VS}}$ is a lifted dependency graph of δ used to identify abstract subproblems, δ is the system of interest, and $\mathsf{c}_{\mathcal{G}_{\mathrm{VS}}}(vs) \equiv \{vs_2 \mid vs_2 \in \mathcal{G}_{\mathrm{VS}} \wedge vs \rightarrow vs_2\}$.

Definition 10 (Acyclic Dependency Compositional Bound)**.**

$$\mathsf{N}\langle b\rangle(vs, \delta, \mathcal{G}_{\mathrm{VS}}) = b(\delta\lfloor_{vs})(1 + \sum_{c \in \mathsf{c}_{\mathcal{G}_{\mathrm{VS}}}(vs)} \mathsf{N}\langle b\rangle(c, \delta, \mathcal{G}_{\mathrm{VS}}))$$

Then, let $\mathsf{N}_{\mathsf{sum}}\langle b\rangle(\delta, \mathcal{G}_{\mathrm{VS}}) = \sum_{vs \in \mathcal{G}_{\mathrm{VS}}} \mathsf{N}\langle b\rangle(vs, \delta, \mathcal{G}_{\mathrm{VS}})$.

Example 4. *For $\mathcal{G}_{\mathrm{VS}}$ from Example 3, we have $\mathsf{N}_{\mathsf{sum}}\langle b\rangle(\delta, \mathcal{G}_{\mathrm{VS}}) = b(\delta\lfloor_{\mathrm{ROOM1}}) + b(\delta\lfloor_{\mathrm{ROOM2}})$.*

Theorem 1. *For an acyclic lifted dependency graph $\mathcal{G}_{\mathrm{VS}}$, if b bounds ℓ, then $\ell(\delta) \leq \mathsf{N}_{\mathsf{sum}}\langle b\rangle(\delta, \mathcal{G}_{\mathrm{VS}})$.*

The previous theorem suggests appropriate choices of b. Taking $b = \ell$ is admissible, while taking b to be the diameter is problematic, as systems exist where $d(\delta) > \mathsf{N}_{\mathsf{sum}}\langle d\rangle(\delta, \mathcal{G}_{\mathrm{VS}})$ (see (Abdulaziz, Gretton, and Norrish 2015) for more details). However, in practice one need not evaluate NP-hard functions, such as ℓ, or the length of the longest path (a.k.a. *recurrence diameter*). Instead b can be an easier to compute function that is an upper bound on ℓ, like the state-space cardinality – i.e. we could take $b(\delta) = 2^{|\mathcal{D}(\delta)|}$, or leverage a more refined cardinality approach as in Rintanen and Gretton (2013).

Exploiting State-Space Acyclicity

The practical utility of dependency graph based decompositions (like $\mathsf{N}_{\mathsf{sum}}$) provides a good motivation to pursue other structures, like state-space acyclicity. In the next example we show that state-space acyclicity is independent of acyclicity in variable dependency. Thus, methods previously developed cannot be used to exploit the former in compositional upper bounding.

Example 5. *$\delta\lfloor_{C_1}$ is acyclic. For example, no state satisfying $CK_{1,2}$ can be reached from a state satisfying $CK_{1,3}$. Now consider $\delta\lfloor_{\mathrm{ROOM1}}$ from Example 1. The dependency graph of $\delta\lfloor_{\mathrm{ROOM1}}$ is comprised of one strongly connected component (SCC). Thus, acyclicity in the assignments of C_1 cannot be exploited in $\delta\lfloor_{\mathrm{ROOM1}}$ by analysing its dependency graph.*

To exploit state-space acyclicity we formalise it as follows.

Definition 11 (Acyclic Transition System)**.** *δ is acyclic iff $\forall x, x' \in \mathbb{U}. x \neq x'$ then $x \not\rightsquigarrow x'$ or $x' \not\rightsquigarrow x$.*

We now investigate how such acyclicity can be used for bounding. Let b be an arbitrary bounding function that satisfies $d(\delta) \leq b(\delta)$ for any δ. Consider a system δ where for some variables vs we have that $\delta\lfloor_{vs}$ is acyclic – i.e. the state-space of $\delta\lfloor_{vs}$ forms a directed acyclic graph (DAG). In that case, we have that $d(\delta) \leq \mathsf{S}_{\mathsf{max}}\langle b\rangle(vs, \delta)$, where $\mathsf{S}_{\mathsf{max}}$ is a compositional bounding function defined as follows.

Definition 12 (Acyclic System Compositional Bound)**.** *Letting $\mathsf{succ}(x, \delta) \equiv \{x' \mid \exists \pi \in \delta. \mathsf{ex}(x, \pi) = x'\}$, S is*

$$\mathsf{S}\langle b\rangle(x, vs, \delta) = b(\delta_{\!\restriction x}) + \max_{x' \in \mathsf{succ}(x, \delta\lfloor_{vs})} (\mathsf{S}\langle b\rangle(x', vs, \delta) + 1)$$

Then, let $\mathsf{S}_{\mathsf{max}}\langle b\rangle(vs, \delta) = \max_{x \in \mathbb{U}(\delta\lfloor_{vs})} \mathsf{S}\langle b\rangle(x, vs, \delta)$.

Theorem 2. *If $\delta\lfloor_{vs}$ is acyclic and b bounds d, then $d(\delta) \leq \mathsf{S}_{\mathsf{max}}\langle b\rangle(vs, \delta)$.*

A formal proof is provided in the next section. S is only well-defined if $\delta\lfloor_{vs}$ is acyclic. We only seek to consider and interpret $\mathsf{S}_{\mathsf{max}}$ in systems $\delta\lfloor_{vs}$ where no execution can visit a state more than once. In that situation $\mathsf{S}_{\mathsf{max}}$ calculates the maximal cost of a traversal through the DAG formed by the state-space of $\delta\lfloor_{vs}$. Completing that intuition, take the cost of visiting a state x to be $b(\delta_{\!\restriction x})$, and the cost of traversing an edge between states to be 1. These ideas are made concrete below, in Example 6. Also, since $\mathsf{S}_{\mathsf{max}}$ follows the scheme of an algorithm that finds the length of the longest path in a DAG, the runtime of a straightforward implementation of it is linear in the size of the state-space of $\delta\lfloor_{vs}$ and the complexity of computing b.

Example 6. *Since $\delta\lfloor_{C_1}$ is acyclic, and $CK_{1,i} \in \mathbb{U}(\delta\lfloor_{C_1})$, then $\mathsf{S}\langle d\rangle(CK_{1,i}, C_1, \delta)$ is well-defined, for $i \in \{1,2,3\}$. Denoting $d(\delta_{\!\restriction CK_{1,i}})$ with $d_{1,i}$ and $\mathsf{S}\langle d\rangle(CK_{1,i}, C_1, \delta)$ with $\mathsf{S}_{1,i}$, we have $\mathsf{S}_{1,3} = d_{1,3}$ because $\mathsf{succ}(CK_{1,3}, \delta\lfloor_{C_1}) = \emptyset$. We also have $\mathsf{S}_{1,2} = d_{1,2} + 1 + S_{1,3} = d_{1,2} + 1 + d_{1,3}$ and $\mathsf{S}_{1,1} = d_{1,1} + 1 + S_{1,2} = d_{1,1} + 1 + d_{1,2} + 1 + d_{1,3} = d_{1,1} + d_{1,2} + d_{1,3} + 2$.*

In closing, it is worth noting that for the above example, the dependency graph of $\delta\lfloor_{\mathrm{ROOM1}}$ is comprised of one SCC. Therefore there is no lifted dependency graph providing further decomposition. Indeed, exploiting acyclicity in dependency between variables alone, one cannot further decompose the subproblem $\delta\lfloor_{\mathrm{ROOM1}}$. We were able to achieve a more fine grained decomposition of that component above, by exploiting state-space acyclicity.

Proof of Theorem 2

A concise statement of our proof requires additional notations. We use the arrow superscript, $\vec{l}$, to indicate that the variable l is a seuquence. We write [] for the empty sequence, and $h :: \vec{l}$ to indicate a sequence with head element h and tail $\vec{l}$. Given two sequences, $\vec{l_1}$ and $\vec{l_2}$, $\vec{l_1}\#\vec{l_2}$ denotes their concatenation.

Proposition 1. *If $\delta\lfloor_{vs}$ is acyclic and $x \in \mathbb{U}(\delta\lfloor_{vs})$, if $x' \in \mathsf{succ}(x, \delta\lfloor_{vs})$, then $b(\delta\lfloor_{x}) + 1 + \mathsf{S}\langle b\rangle(x', vs, \delta) \leq \mathsf{S}\langle b\rangle(x, vs, \delta)$, for a base case function b.*

Definition 13 (Subsystem Trace). *For a state x, action sequence $\vec{\pi}$, and set of variables vs, let $\partial(x, \vec{\pi}, vs)$ be:*

$$\begin{aligned}\partial(x, [], vs) &= [] \\ \partial(x, \pi :: \vec{\pi}, vs) &= \begin{cases} x' :: \partial(x', \vec{\pi}, vs) & \text{if } x\lfloor_{vs} \neq x'\lfloor_{vs} \\ \partial(x', \vec{\pi}, vs) & \text{otherwise}\end{cases}\end{aligned}$$

where $x' = \mathsf{ex}(x, \pi)$.

Proposition 2. *For any x, $\vec{\pi}$, and vs, if $\partial(x, \vec{\pi}, vs) = []$ then: (i) $x\lfloor_{vs} = \mathsf{ex}(x, \vec{\pi})\lfloor_{vs}$ and (ii) there is $\vec{\pi}'$ where $\mathsf{ex}(x, \vec{\pi}) = \mathsf{ex}(x, \vec{\pi}')$ and $|\vec{\pi}'| \leq d(\delta\lfloor_{x\lfloor_{vs}})$.*

Proposition 3. *For two states x and x', a sequence of states $\vec{x}$, a set of variables vs, and an action sequence $\vec{\pi}$, if $\partial(x, \vec{\pi}, vs) = x' :: \vec{x}$, then there are $\vec{\pi}_1$, π and $\vec{\pi}_2$ such that (i) $\vec{\pi} = \vec{\pi}_1\#\pi :: \vec{\pi}_2$, (ii) $\partial(x, \vec{\pi}_1, vs) = []$, (iii) $\mathsf{ex}(\mathsf{ex}(x, \vec{\pi}_1), \pi) = x'$, and (iv) $\mathsf{ex}(x', \vec{\pi}_2) = \mathsf{ex}(x, \vec{\pi})$.*

Proposition 4. *For any x, $\vec{\pi}_1$, $\vec{\pi}_2$, and vs, we have that $\partial(x, \vec{\pi}_1\#\vec{\pi}_2, vs) = \partial(x, \vec{\pi}_1, vs)\#\partial(\mathsf{ex}(x, \vec{\pi}_1), \vec{\pi}_2, vs)$.*

Lemma 1. *For any δ and vs where $\delta\lfloor_{vs}$ is acyclic, $x \in \mathbb{U}(\delta)$, and $\vec{\pi} \in \delta^*$, there is $\vec{\pi}'$ such that $\mathsf{ex}(x, \vec{\pi}) = \mathsf{ex}(x, \vec{\pi}')$ and $|\vec{\pi}'| \leq \mathsf{S}\langle d\rangle(x\lfloor_{vs}, vs, \delta)$.*[2]

Proof. The proof is by induction on $\partial(x, \vec{\pi})$. The base case, $\partial(x, \vec{\pi}) = []$, is trivial. In the step case we have that $\partial(x, \vec{\pi}) = x' :: \vec{x}$ and the induction hypothesis: for any $x^* \in \mathbb{U}(\delta)$, and $\vec{\pi}^* \in \delta^*$ if $\partial(x^*, \vec{\pi}^*) = \vec{x}$ then there is $\vec{\pi}^{*\prime}$ where $\mathsf{ex}(x^*, \vec{\pi}^*) = \mathsf{ex}(x^*, \vec{\pi}^{*\prime})$ and $|\vec{\pi}^{*\prime}| \leq \mathsf{S}\langle d\rangle(x^*\lfloor_{vs})$.

Since $\partial(x, \vec{\pi}) = x' :: \vec{x}$, we have $\vec{\pi}_1, \pi$ and $\vec{\pi}_2$ satisfying the conclusions of Proposition 3. Based on conclusion i, ii, and iii of Proposition 3 and Proposition 4 we have $\partial(x', \vec{\pi}_2) = \vec{x}$. Accordingly, letting x^*, and $\vec{\pi}^*$ from the inductive hypothesis be x', and $\vec{\pi}_2$, respectively, there is $\vec{\pi}_2'$ such that $\mathsf{ex}(x', \vec{\pi}_2) = \mathsf{ex}(x, \vec{\pi}_2')$ and $|\vec{\pi}_2'| \leq \mathsf{S}\langle d\rangle(x'\lfloor_{vs})$.†

From conclusion ii of Proposition 3 and conclusion ii of Proposition 2 there is $\vec{\pi}_1'$ where $\mathsf{ex}(x, \vec{\pi}_1) = \mathsf{ex}(x, \vec{\pi}_1')$ and $|\vec{\pi}_1'| \leq d(\delta\lfloor_{x\lfloor_{vs}})$. Letting $\vec{\pi}' \equiv \vec{\pi}_1'\#\pi :: \vec{\pi}_2'$, from conclusions iii and iv of Proposition 3 and $\dagger$ we have $\mathsf{ex}(x, \vec{\pi}) = \mathsf{ex}(x, \vec{\pi}')$ and $|\vec{\pi}'| \leq d(\delta\lfloor_{x\lfloor_{vs}}) + 1 + \mathsf{S}\langle d\rangle(x'\lfloor_{vs})$.$\ddagger$

Lastly, from conclusion i of Proposition 2 and conclusion ii of Proposition 3 we have $x\lfloor_{vs} = \mathsf{ex}(x, \vec{\pi}_1)\lfloor_{vs} = \mathsf{ex}(x, \vec{\pi}_1')\lfloor_{vs}$ and accordingly $\mathsf{ex}(x\lfloor_{vs}, \pi\lfloor_{vs}) = x'\lfloor_{vs}$. Based on that we have $x'\lfloor_{vs} \in \mathsf{succ}(x\lfloor_{vs}, \delta\lfloor_{vs})$. Then from Proposition 1 and $\ddagger$ we have $|\vec{\pi}'| \leq \mathsf{S}\langle d\rangle(x\lfloor_{vs})$. □

Theorem 2 follows from Lemma 1 and Definitions 4 and 12.

[2]In the proof, the parameters vs and δ, which are common to every occurrence of functions ∂ and S, are omitted, – e.g. we use the shorthand $\mathsf{S}\langle d\rangle(x\lfloor_{vs})$ for $\mathsf{S}\langle d\rangle(x\lfloor_{vs}, vs, \delta)$.

Algorithms for Upper Bounds

Theorem 2 suggests the possibility of compositional upper bounding of the diameter given the presence of acyclicity in a transition system's state-space. We now investigate practical compositional algorithms based on Theorem 2. One straightforward algorithm is the algorithm PUR.

Algorithm 1: PUR(δ)

$S = \mathsf{min}(\{\mathsf{S}_{\mathsf{max}}\langle \text{PUR}\rangle(vs, \delta) \mid vs \in \Omega(\delta)\} \cup \infty)$
if $S = \infty$ **return** UPBND(δ) **else return** S

Theorem 3. *If* UPBND *bounds d, then $d(\delta) \leq$* PUR*(δ)*

In PUR, Ω is an oracle that returns a set of strict subsets of $\mathcal{D}(\delta)$, where $\forall vs \in \Omega(\delta).\delta\lfloor_{vs}$ is acyclic. Since Ω returns strict subsets, the snapshot has fewer variables than the concrete system and accordingly PUR terminates. In PUR the function UPBND provides upper bounds for the diameters of "base-case" problems – i.e. problems that are not further decomposed. Given this assumption and Theorem 2, PUR itself computes valid upper bounds for the diameter of the whole problem.

A main question for a practical implementation of PUR is the choice of Ω. The trivial choice of all strict subsets of $\mathcal{D}(\delta)$ is impractical. A pragmatic solution which we have adopted, is to take the situation that elements in $\mathcal{D}(\delta)$ model individual assignments in the SAS+ model generated using Fast-Downward's preprocessing step (Helmert 2006). Each element in $\Omega(\delta)$ then corresponds to a set of elements from $\mathcal{D}(\delta)$ that model one multi-valued state variable whose domain transition graph is acyclic.

A source of intractability in PUR comes from the min operator. For a full evaluation, $\mathsf{S}_{\mathsf{max}}$ is recursively called as many as $|\Omega(\delta)|!$ times. In practice we only evaluate $\mathsf{S}_{\mathsf{max}}$ on one *arbitrarily* chosen element from $\Omega(\delta)$. Our experimentation never uncovered a problem where a full evaluation of the min, where computationally feasible, produced a better bound. A second source of computational expense comes from the definition of $\mathsf{S}_{\mathsf{max}}$: PUR can be recursively called a number of times that is linear in the size of the state-space of δ. This happens if $\Omega(\delta)$ is a partition of $\mathcal{D}(\delta)$. Although this worst case scenario is contrived, in practice $\Omega(\delta)$ can cover sufficient elements from $\mathcal{D}(\delta)$ to render PUR impractical. This is demonstrated in the following example.

Example 7. *For $\mathcal{D}(\delta)$, Fast-Downward identifies partition $\{c_1, c_2, l_1, l_2, \{g_{1,2}\}, \{g_{1,3}\}, \{g_{1,5}\}, \{g_{1,6}\}, \{g_{2,2}\}, \{g_{2,3}\}, \{g_{2,5}\}\{g_{2,6}\}, \{s_1\}, \{s_2\}\}$ as SAS+ variable assignments. Let $\Omega(\delta)$ denote that set, excluding $\{s_1\}$ and $\{s_2\}$. Note, $\forall vs \in \Omega(\delta)$ we have that $\delta\lfloor_{vs}$ is acyclic. Consequently, we have that* PUR*(δ) evaluates after $\prod_{vs \in \Omega(\delta)} |vs|$ calls to $\mathsf{S}_{\mathsf{max}}$.*

Hybrid Algorithm

We have just observed a situation where PUR can exhibit a runtime that is linear in the size of the state-space. That is favourable compared to exact calculations of diameter, which in our opening remarks we noted to have worse-than-quadratic runtime. Nevertheless this is unacceptable in our factored setting, and we now seek to alleviate this computational burden by applying $\mathsf{S}_{\mathsf{max}}$ to abstract sub-systems obtained using projections that motivated Definition 10. Such abstractions can be significantly smaller than the concrete systems, thus motivating a hybrid approach that can exponentially reduce bound computation times.

Example 8. *Consider applying the approach outlined in Example 3 to compute* PUR *only on the abstractions $\delta\downharpoonright_{\mathrm{ROOM1}}$ and $\delta\downharpoonright_{\mathrm{ROOM2}}$.* $\mathrm{PUR}(\delta\downharpoonright_{\mathrm{ROOM1}})$ *can be evaluated in $\prod_{vs\in\Omega(\delta\downharpoonright_{\mathrm{ROOM1}})}|vs|$ calls to $\mathsf{S}_{\mathsf{max}}$, where $\Omega(\delta\downharpoonright_{\mathrm{ROOM1}})=\{c_1, \mathit{l}_1, \{g_{1,2}\}, \{g_{1,3}\}, \{g_{2,2}\}, \{g_{2,3}\}\}$. The same observation can be made for the evaluation time of* $\mathrm{PUR}(\delta\downharpoonright_{\mathrm{ROOM2}})$. *Thus the product expression in Example 7 is split into a sum if* PUR *is called on projections.*

We now give an upper bounding algorithm, HYB, that combines exploitation of acyclic variable dependency with exploitation of acyclicity in state-spaces.

Algorithm 2: $\mathrm{HYB}(\delta)$

Compute the dependency graph $\mathcal{G}_{\mathcal{D}(\delta)}$ of δ and its SCCs
Compute the lifted dependency graph $\mathcal{G}_{\mathrm{VS}}$
if $2 \leq |\mathcal{G}_{\mathrm{VS}}.V|$ **return** $\mathsf{N}_{\mathsf{sum}}\langle\mathrm{HYB}\rangle(\delta, \mathcal{G}_{\mathrm{VS}})$
else if $\Omega(\delta) \neq \emptyset$ **return** $\mathsf{S}_{\mathsf{max}}\langle\mathrm{HYB}\rangle(ch(\Omega(\delta)), \delta)$
else return $\mathrm{UPBND}(\delta)$

In HYB, ch is an arbitrary choice function. The termination of HYB follows from two facts. Firstly, $\mathsf{N}_{\mathsf{sum}}$ is only called if the lifted dependency graph is not trivial, i.e. $2 \leq |\mathcal{G}_{\mathrm{VS}}.V|$. Accordingly $\mathsf{N}_{\mathsf{sum}}$ will call HYB on projections with strictly smaller domains than the concrete system. Secondly, since Ω returns strict subsets of $\mathcal{D}(\delta)$, $\mathsf{S}_{\mathsf{max}}$ only calls HYB on snapshots with strictly smaller domains than the concrete system.

Note that in HYB, $\mathsf{S}_{\mathsf{max}}$ is only applied to the given transition system δ if there is no non-trivial projection (*i.e.*if $\mathcal{G}_{\mathcal{D}(\delta)}$ has one SCC), and UPBND is applied only to base-cases. Also note that $\mathcal{G}_{\mathcal{D}(\delta)}$ is constructed and analysed with every recursive call to HYB, as snapshotting in earlier calls can remove variable dependencies as a result of removing actions, leading to the breaking of the SCCs in $\mathcal{G}_{\mathcal{D}(\delta)}$, as shown in Example 9.

Example 9. *As shown in Figure 1b, the dependency graph of $\delta\downharpoonright_{\mathrm{ROOM1}}$ has a single SCC, and thus not susceptible to dependency analysis. Taking a snapshot of $\delta\downharpoonright_{\mathrm{ROOM1}}$ at the assignment $CK_{1,2}$ yields a system with one SCC in its dependency graph as well, as shown in Figure 1e. However, taking the snapshot of $\delta\downharpoonright_{\mathrm{ROOM1}}\!\!\restriction_{CK_{1,2}}$ at the assignment $\{\overline{\mathit{l}_{1,1}}, \mathit{l}_{1,2}, \overline{\mathit{l}_{1,3}}\}$, denoted by $LK_{1,2}$, yields a system with an acyclic dependency graph as shown in Figure 1f.*

We prove HYB is sound by proving it is sound for as tight a base function as possible. Then soundness for using UPBND as a base function follows. As discussed above, d cannot be used, because $\mathsf{N}_{\mathsf{sum}}\langle d\rangle$ is not a valid upper bound on d. However, using the sublist diameter ℓ as a base-case function is sound. To prove that, we derive the following.

Theorem 4. *If $\delta\downharpoonright_{vs}$ is acyclic and b bounds ℓ, then $\ell(\delta) \leq \mathsf{S}_{\mathsf{max}}\langle b\rangle(vs, \delta)$.*

This theorem follows from an argument analogous to that provided for Theorem 2, taking ℓ to be d.Using this theorem, and Theorem 1 in (Abdulaziz, Gretton, and Norrish 2015) the validity of HYB as an upper bound on ℓ (and accordingly, the diameter) follows.

Proposition 5. *If* UPBND *bounds ℓ, then $\ell(\delta) \leq \mathrm{HYB}(\delta)$.*

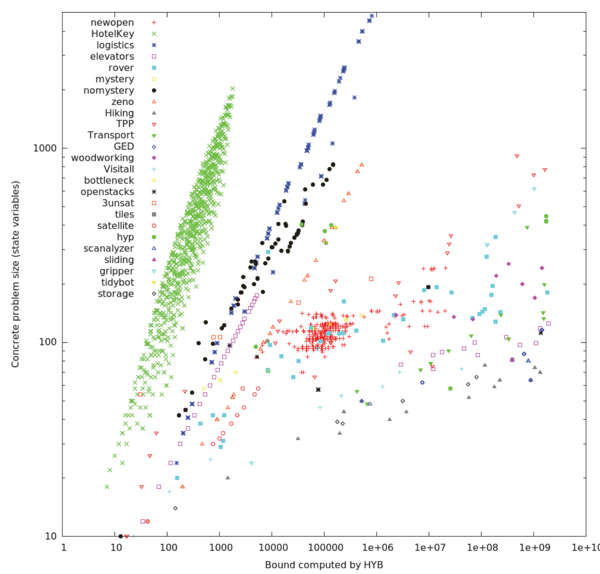


Figure 2: Scatter plot of the bound (horizontal axis) computed by HYB, and the size (i.e. $|\mathcal{D}(\delta)|$) of the concrete problem (vertical).

Empirical Evaluations

We first discuss the practicalities of implementing HYB. Following (Rintanen and Gretton 2013), we take a base-case function, UPBND, which gives the cardinality of the state-space. This choice is pragmatic, taken in light of the fact that computation of alternatives, such as recurrence and sublist diameters, is NP-hard. To optimise computing $\mathsf{N}_{\mathsf{sum}}$ and $\mathsf{S}_{\mathsf{max}}$, we use memoisation, where we compute N or S once for every projection or snapshot, respectively, and store it in a look-up table. This reduced the bound computation time by 70% on average. Our evaluation considers problems from previous International Planning Competitions (IPC), and the unsolvablity IPC, and open Qualitative Preference Rovers benchmarks from IPC2006. Below, the latter are referred to as NEWOPEN.

Quality of HYB Bounds

Two measurements related to a compositional upper bounding algorithm are indicative of its quality. First, we seek an indication of the degree of decompositionality provided by the algorithm. An indication is provided by comparing the size of the domain of the concrete problem—i.e. $|\mathcal{D}(\delta)|$—with that of the largest base-case. A strong decomposition is

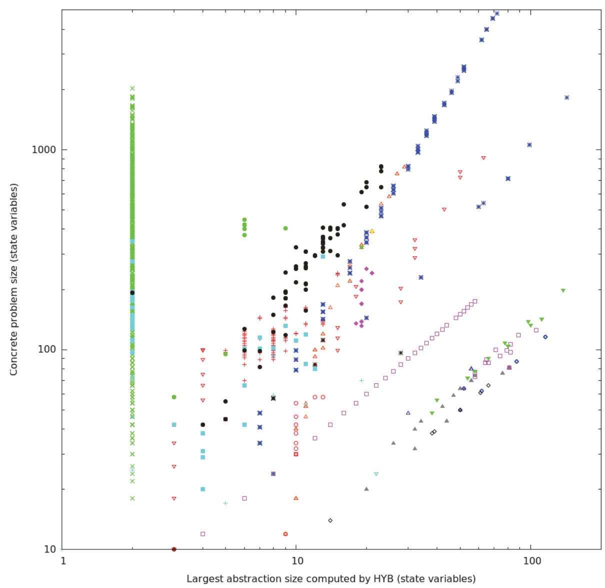


Figure 3: Scatter plot of the size of the largest base-case (horizontal), and the size of the concrete problem (vertical). Legend is provided in Figure 2.

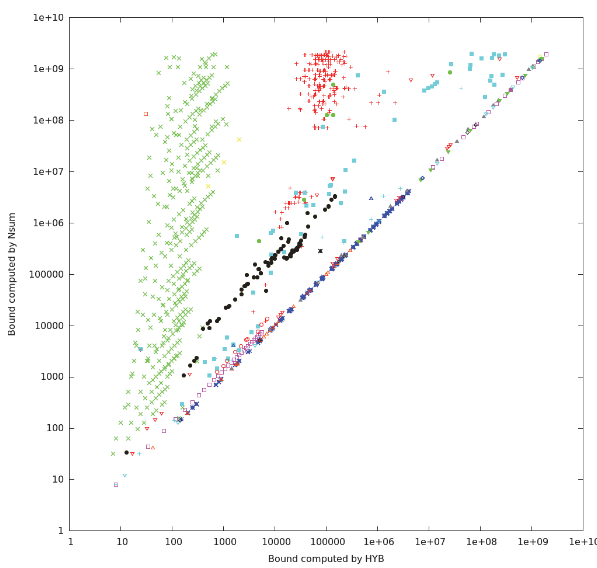


Figure 4: Scatter plot of the bounds computed by HYB (horizontal axis) and the state-of-the-art bounding algorithm $\mathsf{N}_{\mathsf{sum}}$ (vertical). Legend is provided in Figure 2.

indicated when the domain of the base-case is small relative to the concrete problem. Second, we seek an approach that is able to produce bounds that grow sub-exponentiallly with the size of the problem, when they exist. Thus, we measure how the upper bounds scale in domains as the size of the problem instances grow. If the bounds scale gracefully, this indicates an effective compositional approach.

We report our measurements of the performance of HYB in these terms. Our experiments were conducted on a uni-

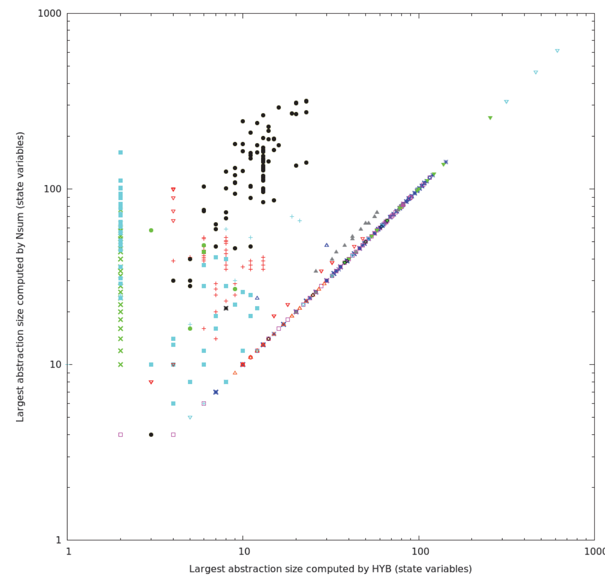


Figure 5: Scatter plot of the size (i.e. $|\mathcal{D}(\delta)|$) of the largest base-case using HYB (horizontal axis) and $\mathsf{N}_{\mathsf{sum}}$ (vertical). Legend is provided in Figure 2.

form cluster with time and memory limits of 30minutes and 4GB, respectively. Figure 3 shows the domain size of the largest base-case compared to the size of the concrete problem. IPC domains with instances remarkably susceptible to decomposition by HYB are: ROVERS (both solvable and unsolvable), STORAGE, TPP (both solvable and unsolvable), LOGISTICS, NEWOPEN, NOMYSTERY (both solvable and over-subscribed), UNSOLVABLE MYSTERY, VISITALL, SATELLITES, ZENO TRAVEL, and ELEVATORS. For those problems, the size of the largest base-case is significantly smaller than the size of the concrete problem, as shown in Figure 3. One IPC domain that is particularly amenable to decomposition is the ROVERS domain, where many of its instances are decomposed to have largest base-cases modelling a single Boolean state-variable. Also, for domains susceptible to decomposition, the bounds computed by HYB grow sub-exponentially with the number of state variables, as shown in Figure 2. We also note that out of those domains, LOGISTICS, NOMYSTERY, SATELLITES, ZENO TRAVEL and ELEVATORS, have linear (or almost linear) growth of the bounds with the size of the problem.

We also ran HYB on a PDDL (Mcdermott et al. 1998) encoding of the hotel key protocol, with the parameters G, k, and R ranging between 1 and 10 (i.e. 1000 instances of the protocol). As shown in Figure 3 (and in the examples earlier), this protocol is particularly amenable to decomposition by HYB. All instances had a largest base-case modelling a single Boolean state-variable. Additionally, the bounds computed by HYB for this set of benchmarks are constant in the number of guests G, grow linearly in the number of rooms R, and quadratically in the number of keys per room K.

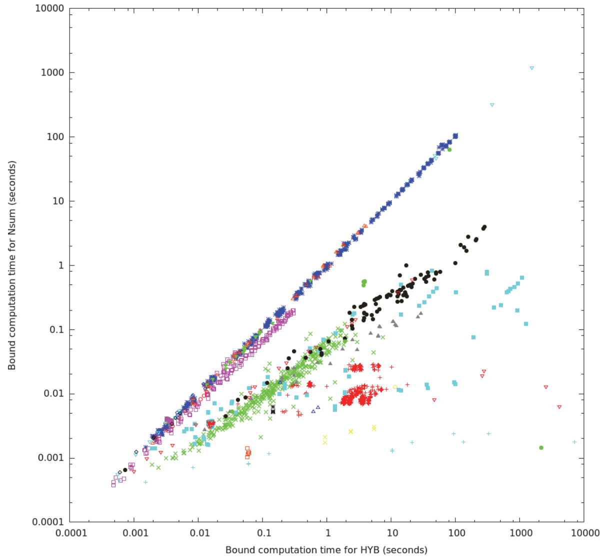


Figure 6: Scatter plot of computation time (in seconds) of HYB (horizontal axis) and $\mathsf{N}_{\mathsf{sum}}$ (vertical) for benchmarks. Legend is provided in Figure 2.

Comparison of HYB and $\mathsf{N}_{\mathsf{sum}}$

We compared the performance of the hybrid compositional bounding algorithm HYB with the state-of-the-art algorithm we refer to as $\mathsf{N}_{\mathsf{sum}}$. The latter was shown in (Abdulaziz, Gretton, and Norrish 2015) to dominate other compositional bounding algorithms. Our experimental cluster and settings are as above. Our analysis and experimentation shows that HYB significantly outperforms $\mathsf{N}_{\mathsf{sum}}$, both in terms of decomposition quality and the tightness of computed bounds. This is particularly the case for the domains: NEWOPEN, NOMYSTERY, ROVERS, HYP, TPP, VISITALL, and BOTTLENECK. The success of HYB in our experimentation reveals something of an abundance of problems with acyclicity in their state-space. Figure 5 indicates that HYB is more successful in decomposing problems compared to $\mathsf{N}_{\mathsf{sum}}$, where the largest base-cases for HYB are smaller than those for $\mathsf{N}_{\mathsf{sum}}$ in 71% of the IPC problems. This observation is reinforced, considering that the 1000th largest bound computed by HYB is $50,534$, while the 1000th largest bound computed by $\mathsf{N}_{\mathsf{sum}}$ is more than 10^6. In the HOTELKEY domain, the difference is even more pronounced. The bound computed by HYB is at most 990 for all the 1000 instances, while for $\mathsf{N}_{\mathsf{sum}}$ only 285 instances have bounds less than 10^6.

Figure 4 shows the computational cost of this improved bounding performance. HYB typically required more computation time than $\mathsf{N}_{\mathsf{sum}}$. However, HYB terminated in 60 seconds, or less, for 93% of the benchmarks. Thus, we have not observed a significant time penalty. We note that the improved decompositionality over $\mathsf{N}_{\mathsf{sum}}$ exhibited here has further application yet to be explored. Should we take UPBND to be a more expensive operator, such as the NP-hard recurrence or sublist diameters, the stronger decomposition indicates that UPBND is invoked for relatively small instances when using HYB compared to $\mathsf{N}_{\mathsf{sum}}$. Thus, computing UPBND can be exponentially easier for decompositions computed by HYB compared to decompositions from $\mathsf{N}_{\mathsf{sum}}$.

Planning with HYB

To evaluate the practical utility of the bounds calculated using HYB, we take them as the queried horizon using the MP version of the SAT-based planner Madagascar (Rintanen 2012). In our experiments we limited the time and memory for planners to 1 hour (inclusive of bound computation) and 4GB. The resulting planner proves the safety of 635 instances of the hotel key protocol, where the instance with 9 rooms, 7 guests, and 45 keys, takes the longest to prove safe – it took just under 30 minutes. This is a substantial improvement over the size of instances automatically proven safe in earlier work. We also ran AIDOS 1 (Seipp et al.) (unsolvability IPC winner) on the hotel key instances and it proved the safety of only 285 of them, where the instance with 2 rooms, 5 guests, and 10 keys, took the longest to prove safe – in 17 minutes. For the IPC benchmarks, our planner proved that 53 instances are unsolvable, 27 of which could not be proven unsolvable by AIDOS 1. The 27 instances are from BOTTLENECK (7 problems), 3UNSAT (4 problems), ELEVATORS (5 problems), and NEWOPEN (11 problems). We also note that compared to the system from (Rintanen and Gretton 2013), we are additionally able to close the heretofore open 7th and 8th Qualitative Preference problem from IPC2006. We also found our bounds useful in solving satisfiable benchmarks. It allowed MP to solve 162 instances that it could not with its default query strategy. Those instances are from ELEVATORS (150 problems), DIAGNOSIS (8 problems), ROVERS (1 problem) and SLIDING-TILES (3 problems).

Conclusions and Future Work

The practical incompleteness of SAT based planning and model-checking algorithms—due to the absence of upper bounding methods—has for some years been noted as a significant problem (Clarke et al. 2004). It is perceived as a deficiency of SAT methods in making comparisons with state based methods. We have addressed that deficiency by advancing the compositional approach to computing upper bounds. Our advance is to exploit state-space acyclicity, giving significantly finer grained decompositions compared to previous works. The resulting algorithm is able to achieve exponentially tighter bounds relative to comparable recent studies. That benefit comes with the risk of an exponential explosion in the number of subproblems considered by the algorithm. The runtime measurements we made experimentally suggest that this theoretical risk is not realised in practice. Bounds computed using our approach enabled a SAT based planning system to prove the unsatisfiability of planning benchmarks (most notably the hotel key protocol) that severely challenge state-of-the-art state-search based tools.

Future research should investigate bounding using functions that are tighter than the diameter. One such bound is the *radius*, which is the longest shortest path from the *initial state* to any other state. We conjecture that our analysis shall carry over to that setting. Further study should also develop compositional bounds using a more sophisticated base-case function. Base-case functions, such as recurrence and sublist diameters could yield superior bounds compared to those we report, however are NP-hard to evaluate. Because the size of abstractions evaluated in the base-case using our method are relatively small compared to other compositional approaches, one can expect exponentially faster bounding using such sophisticated base-case functions.

Acknowledgements We would like to thank Dr. Patrik Haslum and Dr. Alban Grastien for the very useful discussions, suggestions and feedback they gave us, which helped us produce this work.

References

Abboud, A.; Williams, V. V.; and Wang, J. 2016. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete Algorithms*, 377–391. SIAM.

Abdulaziz, M.; Gretton, C.; and Norrish, M. 2015. Verified Over-Approximation of the Diameter of Propositionally Factored Transition Systems. In *Interactive Theorem Proving*. Springer. 1–16.

Aingworth, D.; Chekuri, C.; Indyk, P.; and Motwani, R. 1999. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing* 28(4):1167–1181.

Alon, N.; Galil, Z.; and Margalit, O. 1997. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences* 54(2):255–262.

Baumgartner, J., and Kuehlmann, A. 2004. Enhanced diameter bounding via structural analysis. In *DATE 2004, 16-20 February 2004, Paris, France*, 36–41.

Baumgartner, J.; Kuehlmann, A.; and Abraham, J. 2002. Property checking via structural analysis. In *Computer Aided Verification*, 151–165. Springer.

Biere, A.; Cimatti, A.; Clarke, E. M.; and Zhu, Y. 1999. Symbolic model checking without BDDs. In *TACAS*, 193–207.

Blanchette, J. C., and Nipkow, T. 2010. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In *Interactive Theorem Proving, First International Conference, ITP 2010*, 131–146.

Chan, T. M. 2010. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing* 39(5):2075–2089.

Chechik, S.; Larkin, D. H.; Roditty, L.; Schoenebeck, G.; Tarjan, R. E.; and Williams, V. V. 2014. Better approximation algorithms for the graph diameter. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1041–1052. Society for Industrial and Applied Mathematics.

Clarke, E.; Kroening, D.; Ouaknine, J.; and Strichman, O. 2004. Completeness and complexity of bounded model checking. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, 85–96. Springer.

Fredman, M. L. 1976. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing* 5(1):83–89.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Jackson, D. 2006. *Software Abstractions: Logic, Language, and Analysis*. MIT Press.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2):243–302.

Mcdermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL: The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Nipkow, T. 2006. Verifying a hotel key card system. In Barkaoui, K.; Cavalcanti, A.; and Cerone, A., eds., *Theoretical Aspects of Computing (ICTAC 2006)*, volume 4281 of *Lecture Notes in Computer Science*. Springer. Invited paper.

Rintanen, J., and Gretton, C. O. 2013. Computing upper bounds on lengths of transition sequences. In *International Joint Conference on Artificial Intelligence*.

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artificial Intelligence* 193:45–86.

Roditty, L., and Vassilevska Williams, V. 2013. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 515–524. ACM.

Seipp, J.; Pommerening, F.; Sievers, S.; Wehrle, M.; Fawcett, C.; and Alkhazraji, Y. Fast downward aidos.

Slind, K., and Norrish, M. 2008. A brief overview of HOL4. In *Theorem Proving in Higher Order Logics*, volume 5170 of *LNCS*, 28–32. Springer.

Williams, B. C., and Nayak, P. P. 1997. A reactive planner for a model-based executive. In *International Joint Conference on Artificial Intelligence*, 1178–1185. Morgan Kaufmann Publishers.

Yuster, R. 2010. Computing the diameter polynomially faster than apsp. *arXiv preprint arXiv:1011.6181*.