The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Plan-Length Bounds: Beyond 1-Way Dependency

Mohammad Abdulaziz
Technical University of Munich, Munich, Germany

Abstract

We consider the problem of compositionally computing upper bounds on lengths of plans. Following existing work, our approach is based on a decomposition of state-variable dependency graphs (a.k.a. causal graphs). Tight bounds have been demonstrated previously for problems where key dependencies flow in a single direction—i.e. manipulating variable v_1 can disturb the ability to manipulate v_2 and *not* vice versa. We develop a more general bounding approach which allows us to compute useful bounds where dependency flows in both directions. Our approach is practically most useful when combined with earlier approaches, where the computed bounds are substantially improved in a relatively broad variety of problems. When combined with an existing planning procedure, the improved bounds yield coverage improvements for both solvable and unsolvable planning problems.

Introduction

Many techniques for solving reachability problems in transition systems, like SAT-based planning (Kautz and Selman 1992) and bounded model checking (Biere et al. 1999), benefit from knowledge of *upper bounds* on the lengths of transition sequences, aka *completeness thresholds*. If N is such a bound, and if a transition sequence achieving the goal exists, then that sequence need not comprise more than N actions.

Biere et al. (1999) identify the *diameter* and the *recurrence diameter*, which are topological properties of the state space, as conceptually appealing upper bounds. The diameter is the longest shortest path between any two states. The recurrence diameter is the longest simple path in the state space, i.e. the longest path that does not traverse any state more than once. Approximate and exact algorithms have been developed to calculate both properties given an explicit (e.g. tabular) representation of the transition system. Exact algorithms to compute the diameter 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). The situation is even worse for the recurrence diameter, whose computation is NP-Hard (Pardalos and Migdalas 2004). The impracticality of those explicit algorithms is exacerbated in settings where systems are described using factored representations, like planning and model-checking, since a system's explicit representation can be exponential in the size of the corresponding factored problem description.

However, *compositional* approaches to calculate bounds for problems described using factored representations are feasible. The concrete system's (recurrence) diameter is bounded by composing together bounds for abstract subsystems. The subsystems are *projections* of the concrete system. The motivation for compositional approaches is that they provide useful approximations of plan bounds using smaller computational effort, since only explicit representations of abstract subsystems have to be constructed. Baumgartner, Kuehlmann, and Abraham (2002) and Rintanen and Gretton (2013) followed this projection-based approach to develop procedures to compositionally bound the diameter. This was later improved by Abdulaziz, Gretton, and Norrish (2015) and Abdulaziz, Gretton, and Norrish (2017) (we refer to those two papers as AGN1 and AGN2 hereafter).

A common drawback with previous approaches, however, is that they are only usefully applicable if the projections are induced by acyclicity in the causal/dependency graph (Williams and Nayak 1997; Knoblock 1994). Thus it is an open question as to whether there are practically useful ways to upper-bound plan lengths using projections not resulting from dependency graphs with acyclicity. We address this question by developing a compositional bounding procedure which is applicable to arbitrary projections. We do so by defining a new topological property for the state space to which we refer as the *traversal diameter*. The distinctive feature of the traversal diameter is that, for any given factored transition system, the traversal diameter of that system is upper bounded by the product of the traversal diameters of projections of that system. Those projections can be obtained using any partition of the system's state variables, i.e. the restriction of having acyclic dependencies is not required for this compositional bound.

Using the traversal diameter based compositional method to bound concrete problems is not very practically useful since it uses the product of all the projections' traversal diameters as the bound. However, the utility of this new procedure is most pronounced when combined with previous

state-of-the-art algorithms from AGN1 and AGN2. We use the traversal diameter based method to further decompose "atomic sub-problems", which are abstractions that cannot be further abstracted by the algorithms from AGN1 and AGN2. We experimentally show that this additional decomposition improves the bounds computed by the algorithms from AGN1 and AGN2. For instance, bounds computed using the method from AGN2 improved in 68% of the planning problems on which we experimented, with an improvement of at least 50% in 71% of the cases.

We also use the bounds computed by the new algorithm as horizons for the SAT-based planner Madagascar MP (Rintanen 2012). This improves the coverage of MP in both solvable and unsolvable planning instances compared to the default query strategies or to upper bounds computed by all previous algorithms as horizons.

Background and Notations

Compositional bounds are defined on *factored transition systems* which 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 since the notion of diameter and other bounds are independent of those features.

Definition 1 (States and Actions)**.** *A maplet, $v \mapsto b$, maps a variable v—i.e. a state-characterising proposition—to a Boolean b. A state, x, is a finite set of maplets. 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 $\{v \mapsto b \mid v \in \mathcal{D}(x_1) \cup \mathcal{D}(x_2) \wedge \mathsf{if}\ v \in \mathcal{D}(x_1)\ \mathsf{then}\ b = x_1(v)\ \mathsf{else}\ b = x_2(v)\}$. Note that the state x_1 takes precedence. An action is a pair of states, (p, e), where p represents the* preconditions *and e represents the* effects*. For action $\pi = (p, e)$, the domain of that action is $\mathcal{D}(\pi) \equiv \mathcal{D}(p) \cup \mathcal{D}(e)$.*

Definition 2 (Execution)**.** *When an action $\pi\ (= (p, e))$ is executed at state x, it produces a successor state $\pi(x)$, formally defined as $\pi(x) = \mathsf{if}\ p \not\subseteq x\ \mathsf{then}\ x\ \mathsf{else}\ e \uplus x$. We lift execution to lists of actions $\vec{\pi}$, so $\vec{\pi}(x)$ denotes the state resulting from successively applying each action from $\vec{\pi}$ in turn, starting at x.*

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, 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. $\mathcal{D}(\delta)$ denotes the domain of δ, which is the union of the domains of all the actions it contains. Let $\mathsf{set}(\vec{\pi})$ be the set of elements from $\vec{\pi}$. The set of valid action sequences, δ^*, is $\{\vec{\pi} \mid \mathsf{set}(\vec{\pi}) \subseteq \delta\}$. The set of valid states, $\mathbb{U}(\delta)$, is $\{x \mid \mathcal{D}(x) = \mathcal{D}(\delta)\}$. $\mathcal{G}(\delta)$ denotes the state space of δ, i.e. the digraph whose vertices are $\mathbb{U}(\delta)$ and edges are $\{(x, \pi(x)) \mid x \in \mathbb{U}(\delta), \pi \in \delta\}$.*

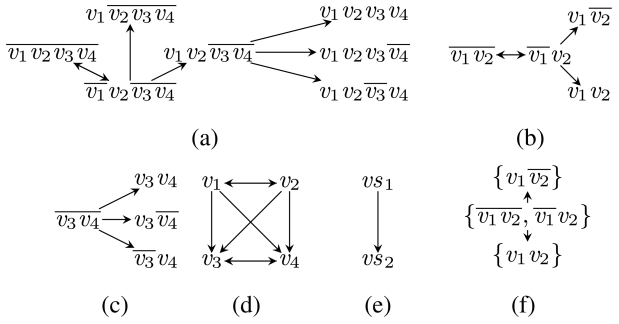

Figure 1: a) is the largest connected component of the state space of δ in Example 1. b) and c) are the state spaces of the projections of δ on $\{v_1, v_2\}$ and $\{v_3, v_4\}$. d) is the dependency graph $\mathcal{G}_{\mathcal{D}(\delta)}$ of δ and e) is a lifted dependency graph $\mathcal{G}_{\mathrm{VS}}$ for δ. f) is a quotient of the state space of $\delta\downarrow_{vs_1}$.

Example 1. *Consider the factored representation, $\delta = \{\pi_1 = (\{\overline{v_1}, \overline{v_2}\}, \{v_2\}), \pi_2 = (\{\overline{v_1}, v_2\}, \{\overline{v_2}\}), \pi_3 = (\{\overline{v_1}, v_2\}, \{v_1, \overline{v_2}\}), \pi_4 = (\{\overline{v_1}, v_2\}, \{v_1\}), \pi_5 = (\{v_1, v_2, \overline{v_3}, \overline{v_4}\}, \{v_4\}), \pi_6 = (\{v_1, v_2, \overline{v_3}, \overline{v_4}\}, \{v_3\}), \pi_7 = (\{v_1, v_2, \overline{v_3}, \overline{v_4}\}, \{v_3, v_4\})\}$. The digraph in Figure 1a represents the largest connected component in the state space of δ, where different states defined on the variables $\mathcal{D}(\delta) = \{v_1, v_2\ v_3, v_4\}$ are shown. Interpreting δ as a transition system, it has two "modes" of operation. The first mode changes the assignments of $\{v_1, v_2\}$ and is represented by actions $\{\pi_1 \cdots \pi_4\}$. The second mode, represented by actions $\{\pi_5, \pi_6, \pi_7\}$, changes the assignments of $\{v_3, v_4\}$ and is only enabled if $\{v_1, v_2\}$ are both set to true.*

Compositional Bounding

For a system δ, a bound on the length of action sequences is $\mathrm{EXP}(\delta) = 2^{|\mathcal{D}(\delta)|} - 1$ (i.e. one less than the size of the state space), where $|\bullet|$ denotes the cardinality of a set or the length of a list. Other bounds employed by previous approaches are topological properties of the state space like the diameter, suggested by Biere et al. 1999.

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

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

Note that if there is a valid action sequence between any two states, then there is a valid action sequence between them which is no longer than the diameter. Another topological property suggested by Biere et al. 1999 as a suitable upper bound is the recurrence diameter.

Definition 5 (Recurrence Diameter)**.** *Let $\mathsf{distinct}(x, \vec{\pi})$ denote that all states traversed by executing $\vec{\pi}$ at x are distinct states. The recurrence diameter is the length of the longest simple path in the state space, formally*

$$rd(\delta) = \max_{x \in \mathbb{U}(\delta), \vec{\pi} \in \delta^*, \mathsf{distinct}(x, \vec{\pi})} |\vec{\pi}|$$

Choosing a topological property as an upper bound depends on many factors. **Size**: for a system δ, $d(\delta)$ can be exponentially smaller than $rd(\delta)$, which in turn can be exponentially smaller than $\text{EXP}(\delta)$ (Biere et al. 1999). **Complexity**: computing $\text{EXP}(\delta)$ is the easiest since it can be done in a time linear in $|\mathcal{D}(\delta)|$, while the diameter can be computed in time at least quadratic in the size of the state space (Aboud, Williams, and Wang 2016), and the recurrence diameter is the hardest as it is NP-Hard in the size of the state space (Pardalos and Migdalas 2004). **Usability**: the diameter is a completeness threshold for SAT-based AI planning and bounded model checking of safety formulae, while the recurrence diameter is a completeness threshold for bounded model checking of fairness formulae (Biere et al. 1999).

In this work, the most relevant trait of a topological property is the ability to compositionally bound it. By that we mean computing an upper bound on the topological property of a given system by composing topological properties of abstractions of that system. A key abstraction concept for compositional reasoning about transition systems is *projection*, which is defined as follows.

Definition 6 (Projection)**.** *Projecting an object (a state x, an action π, a sequence of actions $\overrightarrow{\pi}$ 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 $x\downarrow_{vs}$, $\pi\downarrow_{vs}$, $\overrightarrow{\pi}\downarrow_{vs}$ and $\delta\downarrow_{vs}$ 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 2. *Let $vs_1 = \{v_1, v_2\}$ and $vs_2 = \{v_3, v_4\}$. A partition of the domain of δ from Example 1 is $\{vs_1, vs_2\}$. The projection $\delta\downarrow_{vs_2} = \{\pi_5\downarrow_{vs_2} = (\{\overline{v_3}, \overline{v_4}\}, \{v_4\}), \pi_6\downarrow_{vs_2} = (\{\overline{v_3}, \overline{v_4}\}, \{v_3\}), \pi_7\downarrow_{vs_2} = (\{\overline{v_3}, \overline{v_4}\}, \{v_3, v_4\})\}$. The variables $\{v_1, v_2\}$ were removed from action preconditions and effects, and actions with empty effects were removed. Figures 1b and 1c show the state spaces of the projections $\delta\downarrow_{vs_1}$ and $\delta\downarrow_{vs_2}$.*

Compositional techniques that use projection bound a topological property of a system δ by topological properties of a set of projections of δ. Those projections are done on a partition $vs_{1..n}$ of the set of state variables $\mathcal{D}(\delta)$. For instance, for any $vs_{1..n}$, $\text{EXP}(\delta)$ is equal to (and accordingly bounded by) $\prod_{vs\in vs_{1..n}}(\text{EXP}(\delta\downarrow_{vs}) + 1) - 1$, i.e. the product of the state space sizes of such a set of projections. In contrast to EXP, the (recurrence) diameter cannot generally be bounded by the (recurrence) diameters of projections (Abdulaziz 2017, Chapter 3, Theorems 1 and 2). However, if there are acyclicities in *variable dependencies*, there are methods that compose the recurrence diameters of projections to bound on the concrete system's diameter (Baumgartner, Kuehlmann, and Abraham 2002; Rintanen and Gretton 2013). We now review variable dependency and the most recent of those methods from AGN1.

Acyclicity in variable dependency has been exploited in previous research by reasoning about the *dependency/causal* graph (Williams and Nayak 1997; Knoblock 1994). We formally describe that graph, reviewing precisely what is meant by dependency.

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: (i) $v_1 = v_2$, (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: (i) $vs_1 \cap vs_2 = \emptyset$, and (ii) there are $v_1 \in vs_1$ and $v_2 \in vs_2$, where $v_1{\rightarrow}v_2$.*

Definition 8 (Digraph Quotient)**.** *For a digraph $\mathcal{G}$, and a partition P of its vertices $V(\mathcal{G})$, the quotient $\mathcal{G}/P$ of $\mathcal{G}$ is the digraph with a vertex for each $u \in P$. $\mathcal{G}/P$ has an edge between any $us_1, us_2 \in P$ iff $\mathcal{G}$ has an edge between any $u_1 \in us_1$ and $u_2 \in us_2$.*

Definition 9 (Dependency Graph)**.** *$\mathcal{G}_{\mathcal{D}(\delta)}$ is a dependency graph of δ, if $\mathcal{D}(\delta)$ are its vertices and $\{(u_1, u_2) \mid u_1{\rightarrow}u_2 \wedge u_1, u_2 \in \mathcal{D}(\delta)\}$ are its edges. A quotient, $\mathcal{G}_{\text{VS}}$, of $\mathcal{G}_{\mathcal{D}(\delta)}$ on a partition $vs_{1..n}$ of $\mathcal{D}(\delta)$ is a lifted dependency graph.*

Example 3. *Figure 1d shows the dependency graph $\mathcal{G}_{\mathcal{D}(\delta)}$ of δ from Example 1, where self-loops are omitted. Figure 1e has a lifted dependency graph $\mathcal{G}_{\text{VS}}$ of δ which is the quotient $\mathcal{G}_{\mathcal{D}(\delta)}/\{vs_1, vs_2\}$.*

AGN1used the compositional bounding function $\mathsf{N}_{\mathsf{sum}}$, defined via the recurrence below. δ is the system of interest, $\mathcal{G}_{\text{VS}}$ is a lifted dependency graph of δ used to identify abstract subproblems, and $\mathsf{child}_{\mathcal{G}_{\text{VS}}}(vs)$ denotes the set of children of a set of varaibles vs in $\mathcal{G}_{\text{VS}}$, $\{vs_2 \mid vs_2 \in V(\mathcal{G}_{\text{VS}}) \wedge vs{\rightarrow}vs_2\}$. The functional parameter b bounds projections and we refer to it as the base case function.

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

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

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

Theorem 1. *For any δ with an acyclic lifted dependency graph $\mathcal{G}_{\text{VS}}$, $d(\delta) \leq \mathsf{N}_{\mathsf{sum}}\langle rd\rangle(\delta, \mathcal{G}_{\text{VS}})$.*

Example 4. *Consider $\mathcal{G}_{\text{VS}}$ from Figure 1e, and the base case function b. Let N_i denote $\mathsf{N}\langle b\rangle(vs_i, \delta, \mathcal{G}_{\text{VS}})$ and b_i denote $b(\delta\downarrow_{vs_i})$. We have (i) $\mathsf{N}_2 = b_2$, (ii) $\mathsf{N}_1 = b_1 + b_1b_2$, and (iii) $\mathsf{N}_{\mathsf{sum}}\langle b\rangle(\delta, \mathcal{G}_{\text{VS}}) = b_1 + b_2 + b_1b_2$.*

Since previous methods only apply to dependencies with acyclicity, it is an open question as to whether there is a compositional bound on plan lengths better than $\text{EXP}(\delta)$ for projections not induced by acyclic lifted dependency graphs. In this work we provide a positive answer to that question.

The Traversal Diameter

The traversal diameter is one less than the largest number of states that could be traversed by any path.

Definition 11 (Traversal Diameter)**.** *Let $\mathsf{ss}(x, \overrightarrow{\pi})$ be the set of states traversed by executing $\overrightarrow{\pi}$ at x. The traversal diameter is*

$$td(\delta) = \max_{x\in\mathbb{U}(\delta), \overrightarrow{\pi}\in\delta^*} |\mathsf{ss}(x, \overrightarrow{\pi})| - 1.$$

Intuitively, the traversal diameter is a version of the diameter that instead of selecting shortest action sequences that reach the same final destination, it selects shortest action sequences that traverse the same states en route to the destination. It should also be clear that the traversal diameter is upper bounded by the state space size.

Example 5. *For $\delta\downarrow_{vs_1}$ from Example 2 whose state space is shown in Figure 1b, the traversal diameter is 2.*

The most appealing feature of the traversal diameter is that it is an upper bound on the recurrence diameter (and accordingly the diameter) that can be compositionally bounded with projections from arbitrary partitions of the state variables. In other words, unlike other topological properties, it avoids any conditions on the dependencies between the projections used for bounding, as shown in the theorem below.

Theorem 2. *For a factored representation δ and a partition $vs_{1..n}$ of $\mathcal{D}(\delta)$, $td(\delta) \leq \prod_{vs \in vs_{1..n}}(td(\delta\downarrow_{vs}) + 1) - 1$.*

To prove Theorem 2 we begin by stating the following propositions.

Proposition 1. *For a number k, if for every $x \in \mathbb{U}(\delta)$ and $\vec{\pi} \in \delta^*$, $|\mathsf{ss}(x, \vec{\pi})| \leq k + 1$, then $td(\delta) \leq k$.*

Proposition 2. *For a set of states S, let $S\downarrow_{vs}$ denote $\{x\downarrow_{vs} \mid x \in S\}$. Let $\mathsf{sat\text{-}pre}(x, \vec{\pi})$ denote that preconditions of every action in $\vec{\pi}$ are satisfied, if $\vec{\pi}$ is executed from x. If $\mathsf{sat\text{-}pre}(x, \vec{\pi})$, then $\mathsf{ss}(x, \vec{\pi})\downarrow_{vs} = \mathsf{ss}(s\downarrow_{vs}, \vec{\pi}\downarrow_{vs})$.*

Proposition 3. *For any partition $vs_{1..n}$ of $\mathcal{D}(\delta)$, $|\mathsf{ss}(x, \vec{\pi})| \leq \prod_{vs \in vs_{1..n}} |\mathsf{ss}(x, \vec{\pi})\downarrow_{vs}|$.*

Perhaps Proposition 3 is the least obvious. Its intuitive meaning is that a set of states is a subset of the cartesian product of its own projections, given that the projection is on a partition of the state variables.

Proof of Theorem 2. Consider $x \in \mathbb{U}(\delta)$ and without loss of generality, an action sequence $\vec{\pi} \in \delta^*$ such that $\mathsf{sat\text{-}pre}(x, \vec{\pi})$. From Definition 11, for any vs, $x\downarrow_{vs} \in \mathbb{U}(\delta\downarrow_{vs})$ and $\vec{\pi}\downarrow_{vs} \in \delta\downarrow_{vs}{}^*$, we have $|\mathsf{ss}(x\downarrow_{vs}, \vec{\pi}\downarrow_{vs})| - 1 \leq td(\delta\downarrow_{vs})$. Theorem 2 then follows from Proposition 2, Proposition 3 and Proposition 1. □

Optimality of the Compositional Bound

The bound in Theorem 2 is optimal in the following sense: any sound compositional bounding function that takes as input (i) projections' traversal diameters and (ii) the dependencies between the projections, will produce a bound that is no less than the bound specified in Theorem 2. In other words this bound cannot be improved except by exploiting more structure than that of the variable dependencies.

Since the optimality theorem quantifies over "compositional bounding functions", we first need to discuss how we formulate such functions. One notion we need to introduce is that of *labelled digraphs*, which are digraphs whose vertices have labels. For example, a lifted dependency graph is a graph whose vertices are labelled by sets of state variables. For a labelled digraph $\mathcal{G}_\alpha$ and a vertex u, the label of u is denoted by $\mathcal{G}_\alpha(u)$. Also, we define and image operation for labelled digraphs that effectively changes the vertex labels. In particular, the image $h(\!|\mathcal{G}_\alpha|\!)$ of the function h on the labelled digraph $\mathcal{G}_\alpha$ is a graph that has the same vertices and edges as $\mathcal{G}_\alpha$, but with the label of every vertex u changed from $\mathcal{G}_\alpha(u)$ to $h(\mathcal{G}_\alpha(u))$. In this setting, one can see the lifted dependency graph as a labelled digraph whose vertices are labelled each with a set of state variables.

A compositional bounding function f is a function that takes the projections' traversal diameters and the dependencies between the projections and returns an upper bound on the traversal diameter of the entire system. As arguments to f, projections' traversal diameters and their dependencies are encoded as a labelled digraph, $\mathcal{G}_\mathbb{N}$, in which every vertex is labelled by a natural number. This digraph has one vertex per projection and every edge represents a dependency between two projections. Every vertex is labelled by a natural number that is the traversal diameter of the corresponding projection.

Theorem 3. *For any digraph, $\mathcal{G}_\mathbb{N}$, with natural number labels, there is a factored system δ such that: (i) $\prod_{u \in V(\mathcal{G}_\mathbb{N})}(\mathcal{G}_\mathbb{N}(u) + 1) - 1 \leq td(\delta)$, and (ii) there is a lifted dependency graph $\mathcal{G}_{\mathrm{VS}}$ for δ, such that $\mathcal{G}_\mathbb{N} = \mathfrak{T}(\!|\mathcal{G}_{\mathrm{VS}}|\!)$, where $\mathfrak{T}(vs) = td(\delta\downarrow_{vs})$.*

The proof is made of three main steps. Firstly, for each given projection traversal diameter m (i.e. m is a label of a vertex $u \in V(\mathcal{G}_\mathbb{N})$) we construct a factored system $\mathbb{I}^u$ with traversal diameter m. Those systems are constructed such that: i) their union is a system with a traversal diameter more than $f(\mathcal{G}_\mathbb{N})$, and ii) they are projections of the final construction δ. Secondly, for every dependency from projection $\mathbb{I}^{u_1}$ to $\mathbb{I}^{u_2}$ (i.e. an edge in $\mathcal{G}_\mathbb{N}$), we construct an action that has preconditions from $\mathbb{I}^{u_1}$ and effects from $\mathbb{I}^{u_2}$. Those actions are supposed to not change the state space of the final construction, they only add dependencies corresponding to the edges in $\mathcal{G}_\mathbb{N}$. Thirdly, we show that the union of the constructed projections and the dependency inducing actions is the required witness δ, i.e. its diameter exceeds $f(\mathcal{G}_\mathbb{N})$. Before we start the proof, for system δ and states $x, y \in \mathbb{U}(\delta)$, let $x \rightsquigarrow y$ denote that there is a $\vec{\pi} \in \delta^*$ such that $\vec{\pi}(x) = y$.

Proof. For $u \in V(\mathcal{G}_\mathbb{N})$, let $\mathbb{I}^u$ denote the factored system (i.e. set of actions) $\{(x_0^u, x_i^u) \mid 1 \leq i \leq \mathcal{G}_\mathbb{N}(u)\} \cup \{(x_0^u, x_i^u) \mid 1 \leq i \leq \mathcal{G}_\mathbb{N}(u)\}$. For instance, if for a vertex u, $\mathcal{G}_\mathbb{N}(u) = 3$, the state space of $\mathbb{I}^u$ will look like the one depicted in Figure 2c. Also construct those systems s.t. for $u_1 \neq u_2$ we have $\mathcal{D}(\mathbb{I}^{u_1}) \cap \mathcal{D}(\mathbb{I}^{u_2}) = \emptyset$.

Fix some $u \in V(\mathcal{G}_\mathbb{N})$. Let $\mathcal{S}(\mathbb{I}^u)$ denote the largest connected component in the state space of $\mathbb{I}^u$, which is unique. $x_i^u \rightsquigarrow x_j^u$ holds for any $x_i^u, x_j^u \in \mathcal{S}(\mathbb{I}^u)$, thus $td(\mathbb{I}^u) = |\mathcal{S}(\mathbb{I}^u)| - 1 = \mathcal{G}_\mathbb{N}(u)$.†

Let $\delta = \{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}) \mid (u_1, u_2) \in E(\mathcal{G}_\mathbb{N})\} \cup \bigcup_{u \in V(\mathcal{G}_\mathbb{N})} \mathbb{I}^u$. We now show that δ satisfies requirement (i). Again, let $\mathcal{S}(\delta)$ denote the largest connected component in the state space of δ, which is unique. Since $x_i^u \rightsquigarrow x_j^u$ holds for any $x_i^u, x_j^u \in \mathcal{S}(\mathbb{I}^u)$, then $x \rightsquigarrow y$ holds for any $x, y \in \mathcal{S}(\delta)$, and therefore there is a path that traverses every member of $\mathcal{S}(\delta)$. Since for $u_1 \neq u_2$ we have $\mathcal{D}(\mathbb{I}^{u_1}) \cap$

$\mathcal{D}(\mathbb{I}^{u_2}) = \emptyset$, we have $|\mathcal{S}(\delta)| = \prod_{u \in V(\mathcal{G}_\mathbb{N})} |\mathcal{S}(\mathbb{I}^u)|$. Since from † we have $\prod_{u \in V(\mathcal{G}_\mathbb{N})} |\mathcal{S}(\mathbb{I}^u)| = \prod_{u \in V(\mathcal{G}_\mathbb{N})} (\mathcal{G}_\mathbb{N}(u)+1)$, then $\prod_{u \in V(\mathcal{G}_\mathbb{N})} (\mathcal{G}_\mathbb{N}(u)+1) - 1 \leq td(\delta)$.

To show that δ satisfies requirement (ii), consider a relabelling, $\mathcal{G}_{\text{VS}}$, of $\mathcal{G}_\mathbb{N}$, where every vertex u is relabelled by the domain of the system $\mathbb{I}^u$. Recall that δ had the set of actions $\{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}) \mid (u_1, u_2) \in E(\mathcal{G}_\mathbb{N})\}$ as a subset. These actions are constructed such that they add dependency from $\mathcal{D}(\mathbb{I}^{u_1})$ to $\mathcal{D}(\mathbb{I}^{u_2})$ in δ iff $(u_1, u_2) \in E(\mathcal{G}_\mathbb{N})$. Accordingly edges of $\mathcal{G}_{\text{VS}}$ represent the dependencies of δ and accordingly it is a lifted dependency graph of δ. Also since, for $u \in V(\mathcal{G}_\mathbb{N})$, $\delta|_{\mathcal{D}(\mathbb{I}^u)} = \mathbb{I}^u$, and by construction $\mathcal{G}_\mathbb{N}$ is a relabelling of $\mathcal{G}_{\text{VS}}$, and from † we have $\mathcal{G}_\mathbb{N} = \mathfrak{T}(\!|\mathcal{G}_{\text{VS}}|\!)$. □

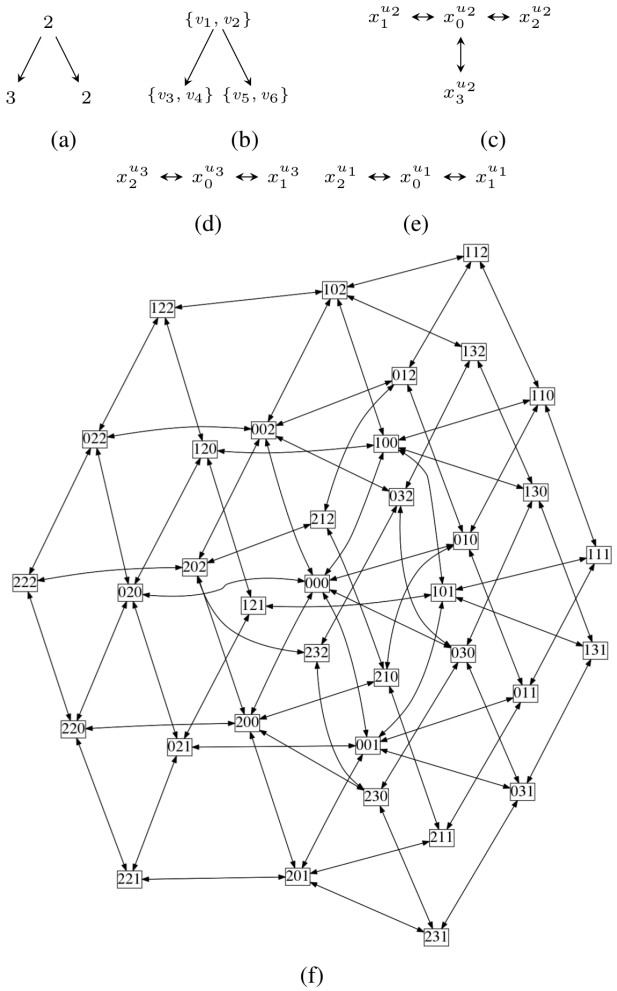

Figure 2: Referring to Example 6, (a) is a natural number labelled graph. (b) a lifted dependency graph of the factored system δ from Example 6. (c), (d), (e) are the largest connected components in the state spaces of the systems $\mathbb{I}^{u_2}$, $\mathbb{I}^{u_3}$ and $\mathbb{I}^{u_1}$, respectively. (f) is the largest connected component in the state space of δ.

Example 6. *This is an example of the construction from Theorem 3, for the natural number labelled digraph $\mathcal{G}_\mathbb{N}$ in Figure 2a. In $\mathcal{G}_\mathbb{N}$ there are three vertices u_1 (the root), u_2, and u_3, labelled by the numbers 2, 3, and 2, respectively. We construct three systems, one per vertex, shown in Figures 2c-2e. For u_2 the constructed system is $\mathbb{I}^{u_2} = \{(x_0^{u_2}, x_1^{u_2}), (x_0^{u_2}, x_2^{u_2}), (x_0^{u_2}, x_3^{u_2}), (x_1^{u_2}, x_0^{u_2}), (x_2^{u_2}, x_0^{u_2}), (x_3^{u_2}, x_0^{u_2})\}$. The states are defined as $x_0^{u_2} = \{\overline{v_3}, \overline{v_4}\}$, $x_1^{u_2} = \{\overline{v_3}, v_4\}$, $x_2^{u_2} = \{v_3, \overline{v_4}\}$, $x_3^{u_2} = \{v_3, v_4\}$. For u_3 the constructed system is $\mathbb{I}^{u_3} = \{(x_0^{u_3}, x_1^{u_3}), (x_0^{u_3}, x_2^{u_3}), (x_1^{u_3}, x_0^{u_3}), (x_2^{u_3}, x_0^{u_3})\}$. The states are defined as $x_0^{u_3} = \{\overline{v_5}, \overline{v_4}\}$, $x_1^{u_3} = \{\overline{v_5}, v_4\}$ and $x_2^{u_3} = \{v_5, \overline{v_4}\}$. For u_1 the constructed system is $\mathbb{I}^{u_1} = \{(x_0^{u_1}, x_1^{u_1}), (x_0^{u_1}, x_2^{u_1}), (x_1^{u_1}, x_0^{u_1}), (x_2^{u_1}, x_0^{u_1})\}$. The states are defined as $x_0^{u_1} = \{\overline{v_1}, \overline{v_2}\}$, $x_1^{u_1} = \{\overline{v_1}, v_2\}$, and $x_2^{u_1} = \{v_1, \overline{v_2}\}$. For any $x_i^{u_1}, x_j^{u_1} \in \mathcal{S}(\mathbb{I}^{u_1})$, $x_i^{u_1} \rightsquigarrow x_j^{u_1}$ holds and accordingly $td(\mathbb{I}^{u_1}) = 2$. Similarly, $td(\mathbb{I}^{u_2}) = 3$ and $td(\mathbb{I}^{u_3}) = 2$.*

The required witness is $\delta = \{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}), (x_0^{u_1} \uplus x_0^{u_3}, x_1^{u_3})\} \cup \mathbb{I}^{u_1} \cup \mathbb{I}^{u_2} \cup \mathbb{I}^{u_3}$, where the actions $\{(x_0^{u_1} \uplus x_0^{u_2}, x_1^{u_2}), (x_0^{u_1} \uplus x_0^{u_3}, x_1^{u_3})\}$ add to δ dependencies equivalent to the edges of $\mathcal{G}_\mathbb{N}$, i.e. the dependencies shown in Figure 2b. Also, in the constructed witness, for all states x_0, $x_1 \in \mathcal{S}(\delta)$ (shown in Figure 2f) $x_0 \rightsquigarrow x_1$ holds, and accordingly $td(\delta) = 35$.

Computing the Traversal Diameter

An important aspect of td is that, unlike the diameter or the recurrence diameter, it can be computed in linear time using Algorithm 1. A principal component of computing the traversal diameter is an algorithm to compute the weight of the "weightiest" path in acyclic digraphs, where vertices are assigned numerical weights. The weight of a path is the sum of weights of all the vertices that it traverses added to the number of edges comprising it. The weightiest path is computed using the recurrence $\mathsf{S}_{\mathsf{max}}$.

Definition 12 (Weighted Longest Path)**.** *For a digraph $\mathcal{G}$, let the function $b : V(\mathcal{G}) \Rightarrow \mathbb{N}$ be a function that assigns a natural number for every vertex in $V(\mathcal{G})$. S is*

$$\mathsf{S}\langle b\rangle(u, \mathcal{G}) = b(u) + \max_{u' \in \mathsf{child}_\mathcal{G}(u)} (\mathsf{S}\langle b\rangle(u', \mathcal{G}) + 1)$$

Then, let $\mathsf{S}_{\mathsf{max}}\langle b\rangle(\mathcal{G}) = \max_{u \in V(\mathcal{G})} \mathsf{S}\langle b\rangle(u, \mathcal{G})$.

$\mathsf{S}_{\mathsf{max}}$ is only well-defined if $\mathcal{G}$ is acyclic. The runtime of $\mathsf{S}_{\mathsf{max}}$ is linear in the size of $V(\mathcal{G})$, if the values of S for different vertices are memoised and assuming that b is at most of linear complexity. Accordingly if we use Tarjan's (Tarjan 1972) algorithm to compute the strongly connected components, the runtime of Algorithm 1 would be linear in the size of the state space of the given system.

Algorithm 1: TRAVD(δ)

SCC := set of strongly connected components of $\mathcal{G}(\delta)$
return $\mathsf{S}_{\mathsf{max}}\langle\mathbb{C}\rangle(\mathcal{G}(\delta)/SCC)$, where $\mathbb{C}(s) = |s| - 1$

Theorem 4. TRAVD$(\delta) = td(\delta)$.

Proof. For notational brevity, let $\mathcal{G} = \mathcal{G}(\delta)/SCC$, and for a strongly connected component scc, $\mathsf{S}(scc) = \mathsf{S}\langle\mathbb{C}\rangle(scc, \mathcal{G})$

and $\mathsf{child}(scc) = \mathsf{child}_{\mathcal{G}}(scc)$. Since $\mathcal{G}$ is a DAG, its vertices can be topologically ordered in a list l_{SCC}.

Firstly, we prove $\textsc{Trav}\mathsf{D}(\delta) \leq td(\delta)$. We show that for any strongly connected component $scc \in \mathcal{G}$ there is an action sequence $\overrightarrow{\pi}_{scc} \in \delta^*$ and a state $x_{scc} \in scc$, such that $\mathsf{S}(scc) \leq |\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc})| - 1$, which from Definitions 11 and 12, implies $\textsc{Trav}\mathsf{D}(\delta) \leq td(\delta)$. We prove this by induction on l_{SCC}. The base case, $l_{SCC} = []$, is straightforward. For the step case $l_{SCC} = scc :: l'_{SCC}$,[1] and for any $scc' \in l'_{SCC}$, there is a state $x' \in scc'$ and an action sequence $\overrightarrow{\pi}' \in \delta^*$ where $\mathsf{S}(scc') \leq |\mathsf{ss}(x', \overrightarrow{\pi}')| - 1$. Since scc is a strongly connected component of states in $\mathcal{G}(\delta)$, then there is $\overrightarrow{\pi}'_{scc} \in \delta^*$ and a state $x_{scc} \in scc$, where $\overrightarrow{\pi}_{scc}$ traverses exactly all the states in scc, if executed at $x_{scc} \in scc$, i.e. $\mathsf{ss}(x_{scc}, \overrightarrow{\pi}'_{scc}) = scc$. We have two cases:

Case 1 ($\mathsf{child}(scc) = \emptyset$)**.** *From Definition 12,* $\mathsf{S}(scc) = |scc| - 1 = |\mathsf{ss}(x_{scc}, \overrightarrow{\pi}'_{scc})| - 1$ *holds for this case. Accordingly the required witness* $\overrightarrow{\pi}_{scc}$ *is the same as* $\overrightarrow{\pi}'_{scc}$*.*

Case 2 ($\mathsf{child}(scc) \neq \emptyset$)**.** *Let* scc_{max} *be a strongly connected component where* $\forall\ scc' \in \mathsf{child}(scc)$. $\mathsf{S}(scc') \leq \mathsf{S}(scc_{\mathsf{max}})$*. Because* $scc_{\mathsf{max}} \in \mathsf{child}(scc)$ *we have* $scc_{\mathsf{max}} \in l'_{SCC}$*, and accordingly from the inductive hypothesis there are* $x_{\mathsf{max}} \in scc_{\mathsf{max}}$ *and* $\overrightarrow{\pi}_{\mathsf{max}} \in \delta^*$ *such that* $\mathsf{S}(scc_{\mathsf{max}}) \leq |\mathsf{ss}(x_{\mathsf{max}}, \overrightarrow{\pi}_{\mathsf{max}})| - 1$*.(†) Also, because* $scc_{\mathsf{max}} \in \mathsf{child}(scc)$ *and since both* scc *and* scc_{max} *are strongly connected components, there must be* $\overrightarrow{\pi}' \in \delta^*$*, where* $\overrightarrow{\pi}'_{scc} \frown \overrightarrow{\pi}'(x_{scc}) = x_{\mathsf{max}}$*.* [2] *We now show that* $\overrightarrow{\pi}_{scc} = \overrightarrow{\pi}'_{scc} \frown \overrightarrow{\pi}' \frown \overrightarrow{\pi}_{\mathsf{max}}$ *is the required witness. First, it is easy to see that* $\mathsf{ss}(x_{scc}, \overrightarrow{\pi}'_{scc}) \cup \mathsf{ss}(x_{\mathsf{max}}, \overrightarrow{\pi}_{\mathsf{max}}) \subseteq \mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc})$*. Since* $scc_{\mathsf{max}} \in \mathsf{child}(scc)$*, then* $\mathsf{ss}(x_{\mathsf{max}}, \overrightarrow{\pi}_{\mathsf{max}})$ *is disjoint with* scc*. Accordingly* $|\mathsf{ss}(x_{scc}, \overrightarrow{\pi}'_{scc})| + |\mathsf{ss}(x_{\mathsf{max}}, \overrightarrow{\pi}_{\mathsf{max}})| \leq |\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc})|$*. From this, (†), and Definition 12 we have* $\mathsf{S}(scc) \leq |\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc})| - 1$.

Secondly, we prove $td(\delta) \leq \textsc{Trav}\mathsf{D}(\delta)$ by showing that for any $scc \in \mathcal{G}$, $x_{scc} \in scc$, and $\overrightarrow{\pi}_{scc} \in \delta^*$, we have $|\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc})| - 1 \leq \mathsf{S}(scc)$. Our proof is again by induction on the list l_{SCC}. The base case, $l_{SCC} = []$, is straightforward. The step case $l_{SCC} = scc :: l'_{SCC}$, and we have that for any $scc' \in l'_{SCC}$, $x' \in scc'$, and $\overrightarrow{\pi}' \in \delta^*$, $|\mathsf{ss}(x', \overrightarrow{\pi}')| - 1 \leq \mathsf{S}(scc')$ holds. We have two cases:

Case 1 ($\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc}) \subseteq scc$)**.** *Since* $\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc}) \subseteq scc$ *then* $|\mathsf{ss}(x_{scc}, \overrightarrow{\pi})| \leq |scc|$*. From Definition 12, we know that* $|scc| - 1 \leq \mathsf{S}(scc)$*, and accordingly* $|\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc})| - 1 \leq \mathsf{S}(scc)$.

Case 2 ($\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc}) \not\subseteq scc$)**.** *Since* $\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc}) \not\subseteq scc$*, then there are* $\overrightarrow{\pi}_{scc}$*,* π*, and* $\overrightarrow{\pi}_{\mathsf{child}}$ *such that: (i)* $\overrightarrow{\pi}_{scc} = \overrightarrow{\pi}'_{scc} \frown \pi :: \overrightarrow{\pi}_{\mathsf{child}}$*, (ii)* $\mathsf{ss}(x_{scc}, \overrightarrow{\pi}'_{scc}) \subseteq scc$*, and (iii) letting* $x_{\mathsf{child}} = \pi(\overrightarrow{\pi}'_{scc}(x_{scc}))$*,* $x_{\mathsf{child}} \in scc_{\mathsf{child}}$ *holds, for some* $scc_{\mathsf{child}} \in \mathsf{child}(scc)$*. Using the same argument as the last case, we have* $|\mathsf{ss}(x_{scc}, \overrightarrow{\pi}'_{scc})| \leq |scc|$*.(*) Since* $x_{\mathsf{child}} \in scc_{\mathsf{child}}$*, and from the inductive hypothesis, we have that* $|\mathsf{ss}(x_{\mathsf{child}}, \overrightarrow{\pi}_{\mathsf{child}})| - 1 \leq \mathsf{S}(scc_{\mathsf{child}})$*. Then using (*) and since* $\mathsf{ss}(x_{\mathsf{child}}, \overrightarrow{\pi}_{\mathsf{child}})$ *and* scc *are disjoint, we have* $|\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc})| - 1 \leq |scc| + \mathsf{S}(scc_{\mathsf{child}})$*. From Definition 12, we have* $|scc| + \mathsf{S}(scc_{\mathsf{child}}) \leq \mathsf{S}(scc)$ *and accordingly* $|\mathsf{ss}(x_{scc}, \overrightarrow{\pi}_{scc})| - 1 \leq \mathsf{S}(scc)$.

□

Example 7. *Consider the projection* $\delta\downarrow_{vs_1}$ *of* δ *from Example 2 as input to Algorithm 1. The first step in Algorithm 1 is to compute the SCCs of the state space* $\mathcal{G}(\delta)$*.* $\mathcal{G}(\delta)$ *is shown in Figure 1b, and it has three strongly connected components, thus* $SCC := \{\{\overline{v_1}\,\overline{v_2}\}, \{\overline{v_1}\,v_2\}, \{v_1\,\overline{v_2}\}, \{v_1\,v_2\}\}$*. Those connected components induce the quotient of the state space shown in Figure 1f. Next, the algorithm computes* $\mathsf{S}_{\mathsf{max}}\langle\mathbb{C}\rangle(\mathcal{G}(\delta)/SCC)$*. We have* $\mathsf{S}\langle\mathbb{C}\rangle(\{v_1\,\overline{v_2}\}) = \mathbb{C}(\{v_1\,\overline{v_2}\}) = 0$*, and* $\mathsf{S}\langle\mathbb{C}\rangle(\{v_1\,v_2\}) = \mathbb{C}(\{v_1\,v_2\}) = 0$*.* $\mathsf{S}\langle\mathbb{C}\rangle(\{\overline{v_1}\,\overline{v_2}, \overline{v_1}\,v_2\}) = \mathbb{C}\{\overline{v_1}\,\overline{v_2}, \overline{v_1}\,v_2\} + \mathsf{max}\{\mathsf{S}\langle\mathbb{C}\rangle(\{v_1\,v_2\}), \mathsf{S}\langle\mathbb{C}\rangle(\{v_1\,\overline{v_2}\})\} + 1 = 1 + 0 + 1 = 2$*. Thus,* $\mathsf{S}_{\mathsf{max}}\langle\mathbb{C}\rangle(\mathcal{G}(\delta)/SCC) = 2$*, which is the traversal diameter of the state space of* $\delta\downarrow_{vs_1}$ *and the value returned by Algorithm 1.*

Tightness of the Traversal Diameter

Having a product of the traversal diameters of all projections as a compositional bound may not seem like a practically helpful bound. However, it is a substantial improvement over what can currently be done in the case when there is not a non-trivial acyclic lifted dependency graph. When given a set of projections without acyclicity in the dependencies between them, existing compositional bounding approaches use the product of the projected state space sizes $\textsc{Exp}(\delta)$ as a bound (Rintanen and Gretton 2013, AGN1, and AGN2). Using the product bound of Theorem 2 is a substantial improvement over that because, as shown in the next theorem, the traversal diameter can be exponentially smaller than the size of state space.

Theorem 5. *There are infinitely many factored systems whose traversal diameters are exponentially smaller (in the number of state variables) than the size of their state spaces.*

Proof. For an arbitrary number $n \in \mathbb{N}$, we construct a system whose state space size is a factor of n more than its traversal diameter. Let x_i, for $0 \leq i \leq n$, be $n + 1$ states. Consider the system $\{(x_0, x_i) \mid 1 \leq i \leq n\}$. The traversal diameter of this system is 1 since, the only possible transitions are from state x_0 to a state x_i, for $1 \leq i \leq n$. However the system's state space has at least $n + 1$ states. □

Example 8. $\delta\downarrow_{vs_2}$ *from Example 2 whose state space is shown in Figure 1c is an example of the above construction with* $n = 3$*. We can take* x_0, x_1, x_2*, and* x_3 *to be* $\overline{v_3}\,\overline{v_4}$*,* $\overline{v_3}\,v_4$*,* $v_3\,\overline{v_4}$*, and* $v_3\,v_4$*, respectively.*

[1] For a list l, $h :: l$ is l with the element h appended to its front.

[2] For two lists l_1 and l_2, $l_1 \frown l_2$ denotes their concatenation.

Practical Bounding Using the Traversal Diameter

In the last section, we laid down a theoretical foundation suggesting that the traversal diameter could be successfully used for compositional upper bounding. A schema for algorithms utilising that theoretical framework to compositionally bound the traversal diameter of a system δ is

$$\textsc{Arb}(\delta) = \mathsf{ch}_{vs_{1..n} \in \mathtt{VS}_{1..n}} \prod_{vs \in vs_{1..n}} (\textsc{TravD}(\delta\downarrow_{vs}) + 1) - 1$$

Above, $\mathtt{VS}_{1..n}$ denotes the set of all partitions of the set of state variables $\mathcal{D}(\delta)$, and ch denotes a function that chooses one partition $vs_{1..n}$ to use for compositional bounding.

To fully specify the bounding algorithm ARB we need to determine the choice of the partition of $\mathcal{D}(\delta)$ using which we obtain the projections. Optimally, the function ch would be instantiated with the function min that would choose the partition which results in the smallest bound. However, since the size of $\mathtt{VS}_{1..n}$ is intractable, min would be an impractical solution. We adopt a practically feasible approach used by AGN2: we take the situation where $\mathcal{D}(\delta)$ models all assignments in the SAS+ model generated using Fast-Downward's preprocessing step (Helmert 2006), and choose ch to return a partition $vs_{1..n}$ s.t. each equivalence class in $vs_{1..n}$ has elements that model all the assignments of exactly one SAS+ state variable.

Now that we have fully specified ARB we compare it to other bounding algorithms. We experimentally evaluate different bounding algorithms on problems from previous International Planning Competitions (IPC), and the unsolvablity IPC, open Qualitative Preference Rovers benchmarks from IPC2006 (to which we refer as NEWOPEN) and the hotel-key protocol verification problem from AGN2. Our experiments were conducted on a uniform cluster with time and memory limits of 30minutes and 8GB, respectively.

The first two columns in Table 1 show that compared to $\mathsf{N}_{\mathsf{sum}}\langle\textsc{Exp}\rangle$, ARB fails to compute bounds tighter than 10^9 in most domains. That is because when there is branching in the dependency graph, $\mathsf{N}_{\mathsf{sum}}$ computes a bound that has additive terms like the ones in Example 4, while on the other hand, ARB always returns a bound that is the product of the projections' traversal diameters.

Now, recall that Theorem 5 predicts the possibility for exponential improvement in the computed bound if, instead of EXP, we use ARB to bound a system. This suggests another utilisation of ARB: to use it as a base case function for $\mathsf{N}_{\mathsf{sum}}$ instead of EXP. This way ARB will be only used to bound projections which cannot be further decomposed by $\mathsf{N}_{\mathsf{sum}}$, i.e. projections whose variable dependencies are strongly connected. Indeed, using ARB as a base case function improves the computed bounds in 71% of the problems, and the improvement is at least 50% in 66% of the cases. The second row in Table 1 gives an overview of the improvement in the bounds computed by $\mathsf{N}_{\mathsf{sum}}\langle\textsc{Arb}\rangle$ compared to $\mathsf{N}_{\mathsf{sum}}\langle\textsc{Exp}\rangle$ for different domains. A more detailed comparison is in the top plot of Figure 3.

The Traversal Diameter and State Space Acyclicity

The construction used in the proof of Theorem 3 suggests that the bounds computed by ARB are better than those computed by EXP only if the projections of the system have acyclicity and "branching" in their state space. In fact the following proposition holds.

Proposition 4. *If $\mathcal{G}(\delta)$ is strongly connected, then $td(\delta) = \textsc{Exp}(\delta)$.*

This begs the question of whether ARB can somehow be combined with the algorithm proposed by AGN2, which also exploits acyclicity in the state space, to compute tighter bounds and better system decompositions. We first review the approach of AGN2. A critical element of their approach is a system abstraction to which they refer as a *snapshot*. It 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 13 (Snapshot)**.** *For states x and x', let $\mathsf{agree}(x, x')$ denote $|\mathcal{D}(x) \cap \mathcal{D}(x')| = |x \cap x'|$, i.e. every variable that is in the domains of both x and x' has the same assignment in x and x'. The snapshot of δ at a state x is*

$$\delta\langle\!\!\downarrow_x = \{(p, e) \mid (p, e) \in \delta \wedge \mathsf{agree}(p, x) \wedge \mathsf{agree}(e, x)\}\downarrow_{\overline{\mathcal{D}(x)}}$$

where $\overline{\mathcal{D}(x)}$ denotes $\mathcal{D}(\delta) \setminus \mathcal{D}(x)$.

Based on snapshots, and given a system δ, a set of variables vs, and a base case function b, AGN2 defined a method $\mathsf{S}_{\mathsf{max}}\langle b\rangle(vs, \delta)$.[3] That method computes the weightiest path in the state space $\mathcal{G}(\delta\downarrow_{vs})$, where the weight of a state x is $b(\delta\langle\!\!\downarrow_x)$. It is only defined if $\mathcal{G}(\delta\downarrow_{vs})$ is acyclic. Combining $\mathsf{S}_{\mathsf{max}}$ and $\mathsf{N}_{\mathsf{sum}}$, AGN2 suggested Algorithm 2 as a hybrid approach to exploit acyclicities in state spaces and dependencies. In HYB, $vs_{1..n}$ is a partition of $\mathcal{D}(\delta)$, and $\mathsf{ac}(vs_{1..n})$ is a member of $vs_{1..n}$ s.t. the projection $\delta\downarrow_{\mathsf{ac}(vs_{1..n})}$ has a non-trivial acyclic state space. HYB interleaves the functions $\mathsf{N}_{\mathsf{sum}}$ and $\mathsf{S}_{\mathsf{max}}$. It only calls $\mathsf{S}_{\mathsf{max}}$ if the given system's dependencies are strongly connected and δ has acyclic projections on members of $vs_{1..n}$. If both $\mathsf{N}_{\mathsf{sum}}$ and $\mathsf{S}_{\mathsf{max}}$ cannot be called, HYB uses EXP as a base case function.

Algorithm 2: $\textsc{Hyb}(\delta, vs_{1..n})$

```
SCC := set of strongly connected components of G_{D(δ)}
G_VS := G_{D(δ)}/SCC
if 2 ≤ |V(G_VS)|
  return N_sum⟨Hyb(•, vs_{1..n})⟩(δ, G_VS)
if ∃vs_i = ac(vs_{1..n})
  vs_i := ac(vs_{1..n})
  return S_max⟨Hyb(•, vs_{1..n} \ {vs_i})⟩(vs_i, δ)
return Exp(δ)
```

Example 9. *From Examples 1 and 2, consider the system δ and the partition $vs_{1..n} = \{vs_1, vs_2\}$ of its state variables. The SCCs of the dependency graph of δ are vs_1*

[3] We overload the same symbol used in Definition 12

and vs_2, and thus the lifted dependency graph $\mathcal{G}_{\text{VS}}$ computed by HYB *is the one in Example 3. Accordingly,* HYB *will return* $\mathsf{N}_{\mathsf{sum}}\langle \text{HYB}(\bullet, vs_{1..n})\rangle(\delta, \mathcal{G}_{\text{VS}})$*. Then we have that* $\mathsf{N}_{\mathsf{sum}}\langle \text{HYB}(\bullet, vs_{1..n})\rangle(\delta, \mathcal{G}_{\text{VS}}) = \text{HYB}_1 + \text{HYB}_2 + \text{HYB}_1\text{HYB}_2$*, where* $\text{HYB}_i = \text{HYB}(\delta\downarrow_{vs_i}, vs_{1..n})$ *for* $i \in \{1, 2\}$*. Since* $\delta\downarrow_{vs_2}$ *has the acyclic state space shown in Figure 1c we have* $\text{HYB}_2 = \mathsf{S}_{\mathsf{max}}\langle \text{HYB}(\bullet, vs_1)\rangle(vs_2, \delta\downarrow_{vs_2}) = 1$*. Since the state space of* $\delta\downarrow_{vs_1}$ *is not acyclic,* $\text{HYB}_1 = \text{EXP}(\delta\downarrow_{vs_1}) = 3$*. Thus,* $\text{HYB}(\delta, vs_{1..n}) = 1+3+1\times 3 = 7$.

Since HYB already exploits state space acyclicity using $\mathsf{S}_{\mathsf{max}}$, the main question now is whether using ARB as a base case function for HYB instead of EXP can improve the computed bounds. The short answer to that question is: yes, bounds computed by HYB using ARB as a base case function are better in 68% of the problems compared to those computed with EXP as a base case, and the improvement is at least 50% in 71% of the cases. The third column of Table 1 and the bottom plot of Figure 3 show a fine-grained comparison between the bounds.

To understand the improvement in the computed bounds, recall that if the dependency graph has one SCC, HYB will pick one vs_i from $vs_{1..n}$, s.t. the state space of $\delta\downarrow_{vs_i}$ is acyclic. Then HYB uses $\mathsf{S}_{\mathsf{max}}$ to decompose δ into multiple abstractions: the projection $\delta\downarrow_{vs_i}$ and the snapshots of δ on the different states in $\mathbb{U}(\delta)$. Then $\mathsf{S}_{\mathsf{max}}$ calls HYB recursively on each of the snapshots of δ, with vs_i removed from $vs_{1..n}$. This is repeated until the state space of the projection on each remaining member of $vs_{1..n}$ is not acyclic. Then the base case function is called to bound the projection on the remaining members of $vs_{1..n}$. However, as shown in the next example, if a projection's state space is not acyclic, its traversal diameter can still be much tighter than the size of its state space. This will lead to much tighter bounds computed by HYB if it uses ARB as a base case function instead of EXP.

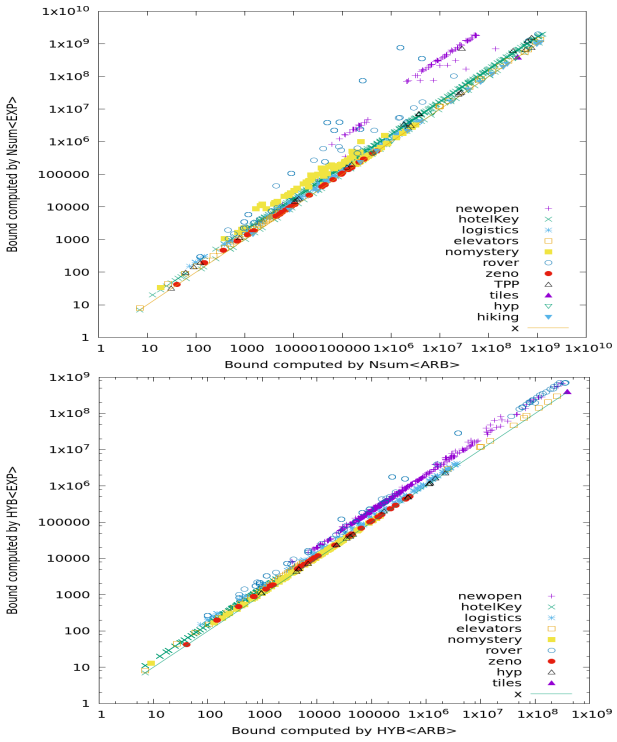

Figure 3: Top (resp. bot.): bounds computed by $\mathsf{N}_{\mathsf{sum}}$ (resp. HYB) when EXP (vert.) is base case function vs ARB (hor.).

Domain(#inst.)	ARB	Nsum				HYB			
newopen(1440)	0	460	1107	459	0	1439	1440	1294	491
hotelKey(1000)	87	518	524	412	412	1000	1000	899	899
logistics(407)	51	407	407	406	365	407	407	406	365
elevators(210)	67	162	163	162	162	162	163	162	162
rover(182)	28	106	119	65	2	176	177	92	6
nomystery(124)	28	124	124	124	51	124	124	124	124
zeno(50)	17	50	50	50	50	50	50	50	50
hiking(40)	20	20	20	20	20	27	26	20	20
TPP(120)	11	53	55	22	21	107	108	19	19
Transport(197)	9	44	43	13	13	45	44	13	13
GED(40)	5	5	5	5	5	5	5	5	5
visitall(90)	16	36	36	16	16	36	36	16	16
bottleneck(50)	6	6	6	6	6	50	50	0	0
openstacks(131)	8	28	28	8	8	116	116	6	6
3unsat(30)	5	5	5	5	5	30	30	0	0
tiles(45)	23	23	23	23	23	23	23	23	23
satellite(10)	10	10	10	10	8	10	10	10	8
hyp(286)	1	187	187	23	23	285	285	33	33
scanalyzer(60)	3	10	10	3	3	10	10	3	3
sliding(25)	13	13	13	13	13	13	13	13	13
gripper(54)	7	39	39	7	7	39	39	7	7
storage(30)	7	7	7	7	7	7	7	7	7
trucks(34)	2	6	6	4	4	7	7	5	5
parcprinter(60)	0	23	23	3	3	27	27	3	3
pipesworld(101)	2	53	53	2	2	55	53	2	2
pegsol(133)	2	3	3	2	2	3	3	2	2

Table 1: Col. 1: the domain name and the number of problems in it. Col. 2: the number of problems for which ARB computed a bound less than 10^9. Col. 3 (resp. 4) has four numbers: (i) problems for which $\mathsf{N}_{\mathsf{sum}}\langle \text{EXP}\rangle$ (resp. $\text{HYB}\langle \text{EXP}\rangle$) computed a bound less than 10^9 (ii) problems for which $\mathsf{N}_{\mathsf{sum}}\langle \text{ARB}\rangle$ (resp. $\text{HYB}\langle \text{ARB}\rangle$) computed a bound less than 10^9 (iii) problems for which the bound by $\mathsf{N}_{\mathsf{sum}}\langle \text{ARB}\rangle$ (resp. $\text{HYB}\langle \text{ARB}\rangle$) is less than the bound by $\mathsf{N}_{\mathsf{sum}}\langle \text{EXP}\rangle$ (resp. $\text{HYB}\langle \text{EXP}\rangle$) (iv) problems for which the bound by $\mathsf{N}_{\mathsf{sum}}\langle \text{ARB}\rangle$ (resp. $\text{HYB}\langle \text{ARB}\rangle$) is less than half the bound by $\mathsf{N}_{\mathsf{sum}}\langle \text{EXP}\rangle$ (resp. $\text{HYB}\langle \text{EXP}\rangle$).

Example 10. *Consider the computation in Example 9. If* ARB *is used as a base case function,* $\text{HYB}_1 = \text{ARB}(\delta\downarrow_{vs_1}, vs_{1..n}) = \text{TRAVD}(\delta\downarrow_{vs_1}) = 2$ *(for the evaluation of* $\text{TRAVD}(\delta\downarrow_{vs_1})$ *see Example 7). Thus* $\text{HYB}(\delta, vs_{1..n}) = 1+2+1\times 2 = 5$.

Using the Bounds To Compute Plans AGN2 showed that when the bounds computed by HYB, with EXP as a base case function, are used as a horizon for the SAT-based planner Madagascar MP (Rintanen 2012), the coverage of MP substantially increased. Indeed, it solved satisfiable and un-

satisfiable planning problems that other state-of-the-art planners could not solve. We now study the improvement in the coverage of MP if we use as horizons the bounds computed by HYB when ARB is the base case. Compared to using EXP as a base case, ARB increases the coverage by 234 for solvable problems. Those problems come from the domains: NEWOPEN (221 problems), ROVER (7 problems), SATELLITE (5 problems), and TPP (1 problem). Also, using ARB as a base case for HYB allows MP to prove the unsolvability of an additional 4 problems from the domain NEWOPEN and 52 problems from the domain HOTELKEY, that it could not solve when EXP is used as a base case function.

Conclusions and Future Work

We contributed a novel compositional upper bounding approach in planning. Our technique exposes problems with a relatively wide variety of dependency structures to upper bounding. Previous approaches only apply to a limited class of problems that have a branching 1-way state variable dependency structure. Our analysis treats a much broader class of problems, with 2-way dependencies. Our new approach, however, is most useful when combined with other existing compositional bounding techniques, where it leads to substantial improvement in the computed bounds. We use it to decompose problem abstractions produced using the other compositional bounding techniques when those abstractions have bidirectional dependencies.

An open problem is to devise a method to practically decompose large concrete problems with strongly connected dependencies instead of only small abstractions produced by other compositional algorithms. Also, investigating the effect of the partition of the state variables used to decompose problems on the value of computed bounds is an interesting avenue for future research.

Acknowledgements We thank Dr. Charles Gretton, Dr. Michael Norrish, Lars Hupel and Simon Wimmer for proofreading parts of this paper, and Lars Hupel for helping me set up the experiments. We also thank Prof. Tobias Nipkow and the German Research Foundation for funding that facilitated this work through the DFG Koselleck Grant NI 491/16-1.

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.

Abdulaziz, M.; Gretton, C.; and Norrish, M. 2017. A State Space Acyclicity Property for Exponentially Tighter Plan Length Bounds. In *International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI.

Abdulaziz, M. 2017. *Formally Verified Compositional Algorithms for Factored Transition Systems*. The Australian National University.

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.; 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.

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.

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.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *ECAI*, 359–363.

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

Pardalos, P. M., and Migdalas, A. 2004. A note on the complexity of longest path problems related to graph coloring. *Applied Mathematics Letters* 17(1):13–15.

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.

Tarjan, R. 1972. Depth-first search and linear graph algorithms. *SIAM Journal on Computing* 1(2):146–160.

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*.