Heuristic Search for the Analysis of Graph Transition Systems

Stefan Edelkamp[1], Shahid Jabbar[1] and Alberto Lluch Lafuente[2]

[1] Computer Science Department, University of Dortmund, Dortmund, Germany
{stefan.edelkamp,shahid.jabbar}@cs.uni-dortmund.de
[2] via del Giardino A 58, I-50053 Empoli, Italy
lafuente@di.unipi.it

Abstract. Graphs are suitable modeling formalisms for software and hardware systems involving aspects such as communication, object orientation, concurrency, mobility and distribution. State spaces of such systems can be represented by graph transition systems, which are basically transition systems whose states and transitions represent graphs and graph morphisms. Heuristic search is a successful Artificial Intelligence technique for solving exploration problems implicitly present in games, planning, and formal verification. Heuristic search exploits information about the problem being solved to guide the exploration process. The main benefits are significant reductions in the search effort and the size of solutions. We propose the application of heuristic search for the analysis of graph transition systems. We define algorithms and heuristics and present experimental results.

1 Introduction

Graphs are a suitable formalism for software and hardware systems involving issues such as communication, object orientation, concurrency, distribution and mobility. The graphical nature of such systems appears explicitly in approaches like graph transformation systems [29] and implicitly in other modeling formalisms like algebras for communicating processes [25]. The properties of such systems mainly regard aspects such as temporal behavior and structural properties. They can be expressed, for instance, by logics used as a basis for a formal verification method, like model checking [5], whose success is mainly due to the ability to find and report errors.

Finding and reporting errors in model checking and many other analysis problems can be reduced to state space exploration problems. In most cases, the main drawback is the *state explosion problem*. In practice, the size of state spaces can be large enough (even infinite) to exhaust the available space and time resources. Heuristic search has been proposed as a solution in many fields, including model checking [14], planning [2] and games [23]. Basically, the idea is to apply algorithms that exploit the information about the problem being solved in order to guide the exploration process. The benefits are twofold: the search effort is reduced, i.e., errors are found faster and by consuming less memory, and the solution quality is improved, i.e., counterexamples are shorter and thus may be more useful. In cases like wide area networks with Quality of Service (QoS), one might not be interested in short paths, but in cheap or optimal ones according to a certain notion of *cost*. Typical examples are algebra defined on Reals, on Booleans, on Probabilities, or on any other system. To cover such a diversity, we generalize our approach by considering an abstract notion of cost.

Our work is mainly inspired by approaches to directed model checking [14], logics for graphs (like the *monadic second order logic* [8]), spatial logics used to reason about the behavior and structure of processes calculi [3] and graphs [4], approaches for the analysis of graph transformation systems [1, 16, 27, 31], and cost-algebraic search algorithms [13, 30].

In another work[1], we suggested a translation of graph transition systems into inputs for action planning. The encoding relates to [11] that compiled protocol software model checking domains in Promela to PDDL. Two of such domains have served as a benchmark for the 4th international planning competition in 2004[2].

The goal of our approach is to formalize a framework for the application of heuristic search in order to analyze structural properties of systems modeled by graph transition systems. We believe that our work additionally illustrates the benefits of applying heuristic search for state space exploration. Heuristic search is intended to reduce the analysis effort and, in addition, to deliver shorter solutions, which in our case means shorter paths in graph transition systems.

Section 2 introduces a running example that is used along the paper to illustrate some of the concepts and methods. Section 3 defines our modeling formalism, namely graph transition systems. Section 4 defines the kind of properties we are interested in verifying. Section 5 summarizes the analysis algorithms and discusses their correctness. Section 6 proposes heuristics for the analysis of properties in graph transition systems. Abstraction is one of the most successful techniques in model checking. In Section 7, we discuss the role of abstraction to define useful heuristic estimates. Section 8 presents experimental results. Section 9 concludes the paper and outlines future research avenues. For the sake of readability proofs are included in an appendix that follows the bibliography.

2 The Arrow Distributed Directory Protocol

The *arrow distributed directory protocol* [9] is a solution to ensure exclusive access to mobile objects in a distributed system. The protocol induces a *distributed queue* structure on a distributed system. The distributed system is given as an undirected graph G, where vertices and edges respectively represent nodes and communication links. Costs are associated with the links in the usual way, and a mechanism for optimal routing is assumed.

[1] Action Planning for graph transition systems, in *ICAPS-Workhop on Verification and Validation of Planning and Scheduling*, 2005. Unarchived manuscript.

[2] `http://ipc.icaps-conference.org/`

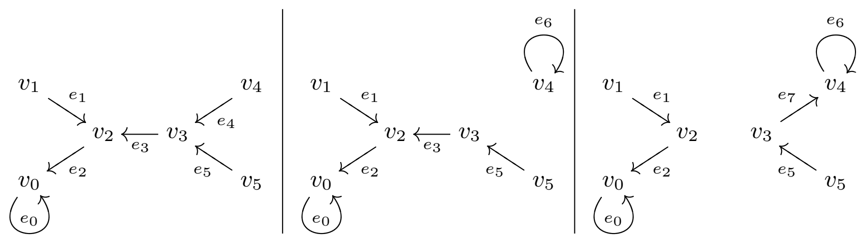

Fig. 1. Three states of the directory.

The protocol works with a minimal spanning tree T of G. Each node has an arrow which either indicates the *direction* in which the object lies or is going to be. If a node owns the object or is requesting it, the arrow points to itself; we say that the node is a *terminal*. The directed graph induced by the arrows is called $\mathcal{L}$. The protocol works by propagating requests and updating arrows such that at any moment the paths induced by arrows, called *arrow paths*, lead to a terminal either owning the object or waiting for it.

Figure 1 illustrates three states of a protocol instance with six nodes $v_0,\ldots,v_5$. For the sake of simplicity only $\mathcal{L}$ is depicted. The state on the left is the initial one: node v_0 has the object and all paths in $\mathcal{L}$ lead to it. The state on the right of the figure is the result of two steps: 1) node v_4 sends a request for the object through its arrow; and 2) v_3 processes it, making its arrow points to v_4. Request propagation should end by making all paths in $\mathcal{L}$ pointing towards v_4, where the object will be transfered once v_0 is finished with it. Each propagation step comprises of two transitions: deleting its out-going edge, and adding the new edge in the direction, where the request came from.

One could be interested in properties like: *Can a certain node v be terminal?* (Property 1), *Can a certain node v be terminal and all arrow paths end at v?* (Property 2), *Can some node be terminal?* (Property 3), *Can some node be terminal and all arrow paths end at it*? (Property 4).

3 Graph Transition Systems

This section presents our algebraic notion of costs. It shall be used as an abstraction of costs or weights associated to edges of graphs or transitions of transition systems. For a deeper treatment of the cost algebra, we refer to [13].

Definition 1 (cost algebra). *A* cost algebra *is a 5-tuple $\langle A, \times, \preceq, \mathbf{0}, \mathbf{1}\rangle$, such that 1) $\langle A, \times\rangle$ is a monoid with $\mathbf{1}$ as identity element and $\mathbf{0}$ as its absorbing element, i.e., $a \times \mathbf{0} = \mathbf{0} \times a = \mathbf{0}$; 2) $\preceq \subseteq A \times A$ is a total ordering with $\mathbf{0} = \bigsqcap A$ and $\mathbf{1} = \bigsqcup A$; A is* isotone, *i.e., $a \preceq b$ implies both $a \times c \preceq b \times c$ and 3) $c \times a \preceq c \times b$ for all $a, b, c \in A$ [30].*

In the rest of the paper $a \prec b$ abbreviates $a \preceq b$ and $a \neq b$. Moreover, $a \succeq b$ abbreviates $b \preceq a$, and $a \succ b$ abbreviates $a \succeq b$ and $a \neq b$.

Intuitively, A is the domain set of cost values, which is linearly ordered by $\preceq$ and has $\sqcup$, $\sqcap$ as least and greatest operations, and $\times$ is the operation used to cumulate values. Consider, for example, the following instances of cost algebras, typically used as cost or QoS formalisms: $\langle\{\mathit{true}, \mathit{false}\}, \wedge, \Rightarrow, \mathit{false}, \mathit{true}\rangle$ (Network and service availability), $\langle\mathbb{R}^+ \cup \{+\infty\}, +, \leq, +\infty, 0\rangle$ (Price, propagation delay) or $\langle\mathbb{R}^+ \cup \{+\infty\}, \min, \geq, 0, +\infty\rangle$ (Bandwidth). In the rest of the paper, we consider a fixed cost algebra $\langle A, \times, \preceq, \mathbf{0}, \mathbf{1}\rangle$.

We now define edge-weighted graphs.

Definition 2 (graph). *A* graph *G is a tuple $\langle V_G, E_G, src_G, tgt_G, \omega_G\rangle$ where V_G is a set of nodes, E_G is a set of edges, $src_G, tgt_G : E_G \to V_G$ are source and target functions, and $\omega_G : E_G \to A$ is a weighting function.*

Graphs usually have a distinguished start state which we denote with s_0^G, or just s_0 if G is clear from the context.

Definition 3 (paths). *A* path *in a graph G is an alternating sequence of nodes and edges represented as $u_0 \overset{e_0}{\to} u_1 \ldots$ such that for each $i \geq 0$, we have $u_i \in V_G$, $e_i \in E_G$, $src_G(e_i) = u_i$ and $tgt_G(e_i) = u_{i+1}$, or, shortly $u_i \overset{e_i}{\to} u_{i+1}$.*

An initial path is a path starting at s_0^G. Finite paths are required to end at states. The length of a finite path p is denoted by $|p|$. The concatenation of two paths p, q is denoted by pq, where we require p to be finite and end at the initial state of q. The cost of a path is the cumulative cost of its edges. Formally,

Definition 4 (paths costs). *Let $p = u_0 \overset{e_0}{\to} \ldots \overset{e_{k-1}}{\to} u_k$ be a finite path in a graph G. The* path cost *$\omega_G(p)$ is $\omega_G(e) \times \omega_G(q)$ if $p = (u \overset{e}{\to} v)q$ and $\mathbf{1}$ otherwise.*

Let $\gamma(u)$ denote the set of all paths starting at node u. In the sequel, we shall use $\omega_G^*(u, V)$ to denote the cost of the optimal path starting at a node u and reaching a node v in a set $V \subseteq V_G$. For ease of notation, we write $\omega_G^*(u, \{v\})$ as $\omega_G^*(u, v)$.

Graph transition systems are suitable representations for software and hardware systems and extend traditional transition systems by relating states with graphs and transitions with partial graph morphisms. Intuitively, a partial graph morphism associated to a transition represents the relation between the graphs associated to the source and the target state of the transition, i.e., it models the merging, insertion, addition and renaming of graph items (nodes or edges), where the cost of merged edges is the least one amongst the edges involved in the merging.

Definition 5 ((partial) graph morphism). *A* graph morphism *$\psi : G_1 \to G_2$ is a pair of mappings $\psi_V : V_{G_1} \to V_{G_2}$, $\psi_E : E_{G_1} \to E_{G_2}$ such that we have $\psi_V \circ src_{G_1} = src_{G_2} \circ \psi_E$, $\psi_V \circ tgt_{G_1} = tgt_{G_2} \circ \psi_E$, and for each $e \in E_{G_2}$ we have, $\omega_{G_2}(e) = \bigsqcup\{\omega_{G_1}(e') \mid \psi_E(e') = e\}$.*
A graph morphism $\psi : G_1 \to G_2$ is called injective *if so are ψ_V and ψ_E;* identity *if both ψ_V and ψ_E are identities, and isomorphism if both ψ_E and ψ_V are*

bijective. A graph G' is a subgraph *of graph G, if $V_{G'} \subseteq V_G$ and $E_{G'} \subseteq E_G$, and the inclusions form a graph morphism.*

A partial graph morphism *$\psi : G_1 \rightarrow G_2$ is a pair $\langle G'_1, \psi_m \rangle$ where G'_1 is a subgraph of G_1, and $\psi_m : G'_1 \rightarrow G_2$ is a graph morphism.*

The composition of (partial) graph morphisms results in (partial) graph morphisms. Now, we define a notion of transition system that enriches the usual ones with weights.

Definition 6 (transition system). *A* Transition System *is a graph $M = \langle S_M, T_M, in_M, out_M, \omega_M \rangle$ whose nodes and edges are respectively called* states *and* transitions*, with in_M, out_M representing the source and target of an edge respectively.*

Finally, we are ready to define graph transition systems, which are transition systems together with morphisms mapping states into graphs and transitions into partial graph morphisms.

Definition 7 (graph transition system). *A* Graph Transition System *(GTS) is a pair $\langle M, g \rangle$, where M is a weighted transition system and $g : M \rightarrow \mathcal{U}(\boldsymbol{G}_p)$ is a graph morphism from M to the graph underlying $\boldsymbol{G}_p$, the category of graphs with partial graph morphisms. Therefore $g = \langle g^S, g^T \rangle$, and the component on states g^S maps each state $s \in S_M$ to a graph $g^S(s)$, while the component on transitions g^T maps each transitions $t \in T_M$ to a partial graph morphism $g^T(t) : g^S(in_M(t)) \Rightarrow g^S(out_M(t))$.*

In the rest of the paper, we shall consider a GTS $\langle M, g \rangle$ modeling the state space of our running example, where g maps states to $\mathcal{L}$, i.e., the graph induced by the arrows, and transitions to the corresponding partial graph morphisms. Consider Figure 1, each of the three graphs depicted, say G_1, G_2 and G_3 corresponds to three states s_1,s_2,s_3, meaning that $g(s_1) = G_1$, $g(s_2) = G_2$ and $g(s_3) = G_3$. The figure illustrates a path $s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3$, where $g(t_1)$ is the identity restricted to all items but edge e_4. Similarly, $g(t_2)$ is the identity restricted to all items but edge e_3. Thus, in both transitions all other items are preserved (with their identity) except the edges mentioned.

4 Properties of Graph Transition Systems

The properties of a graph transition system can be expressed using different formalisms. One can use, for instance, a temporal graph logic like the ones proposed in [1, 27], which combine temporal and graph logics. A similar alternative are spatial logics [3], which combine temporal and structural aspects. In graph transformation systems [7], one can use rules to find certain graphs: the goal might be to find a match for a certain transformation rule. For the sake of simplicity, however, we consider that the problem of satisfying or falsifying a property is reduced to the problem of finding a set of *goal states* characterized by a goal graph and the existence of an injective morphism.

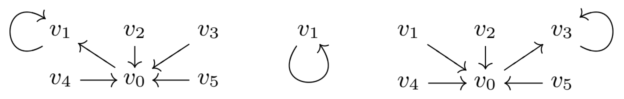

Fig. 2. Three graphs illustrating various goal criteria.

Definition 8 (goal function). *Given a GTS $\langle M, g \rangle$ and a graph G, the* goal function $\text{goal}_G : S_M \to \{\text{true}, \text{false}\}$ *is defined such that* $\text{goal}_G(s) = \text{true}$ *iff there is an injective graph morphism $\psi : G \to g(s)$.*

Intuitively, *goal*$_G$ maps a state s to *true*, if and only if G can be injectively matched with a subgraph of $g(s)$. It is worth mentioning that most graph transformations approaches consider injective rules, for which a match is precisely given by injective graph morphisms, and that the most prominent graph logic, namely the Monadic Second-Order (MSO) logic by Courcelle [8] and its first-order fragment (FO) can be used to express injective graph morphisms. The graph G will be called *goal graph*. It is of practical interest identifying particular cases of goal functions as the following goal types:

1. ψ is an identity - the exact graph G is looked for. In our running example, this corresponds to Property 2 mentioned in Section 2. For instance, we look for the exact graph depicted to the left of Figure 2.
2. ψ is a restricted identity - an exact subgraph of G is looked for. This is precisely Property 1. For instance, we look for a subgraph of the graph depicted to the left of Figure 2. The graph in the center of Figure 2 satisfies this.
3. ψ is an isomorphism - a graph isomorphic to G is looked for. This is precisely Property 4. For instance, we look for a graph isomorphic to the one depicted to the left of Figure 2. The graph to the right of Figure 2 satisfies this.
4. ψ is any injective graph morphism - we have the general case. This is precisely Property 3. For instance, we look for an injective match of the graph depicted in the center of Figure 2. The graph to the right of Figure 2 satisfies this.

Note that there is a type hierarchy, since goal type 1 is a subtype of goal types 2 and 3, which are subtypes of the most general goal type 4.

The computational complexity of the goal function varies according to the above cases. For goals of type 1 and 2, the computational efforts needed are just $O(|G|)$ and $O(|\psi(G)|)$, respectively. Unfortunately, for goal types 3 and 4, due to the search for isomorphisms, the complexity increases to a term exponential in $|G|$ for the graph isomorphism case and to a term exponential in $|\psi(G)|$ for the subgraph isomorphism case. The general problem of subgraph isomorphism (SI) can be reduced polynomially to graph isomorphism. Subgraph isomorphism is NP-complete, as CLIQUE $\leq_p$ SI. The general problem of graph isomorphism is not completely classified. It is expected not to be NP-complete [32].

Now we state the two analysis problems we consider. The first one consists of finding a goal state.

Definition 9 (reachability problem). *Given a GTS $\langle M, g\rangle$ and a graph G (the goal graph), the* reachability problem *of our approach consists of finding a state $s \in S_M$ such that* $\mathrm{goal}(s)$ *is* true*.*

The second problem aims at finding an optimal path to a goal state.

Definition 10 (optimality problem). *Given a GTS (M, g) and a graph G (the goal graph), the* optimality problem *of our approach consists of finding a finite initial path p ending at a state $s \in S_M$ such that such that* $\mathrm{goal}_G(s)$ *is* true *and $\omega(p) = \omega^*_M(s^M_0, S')$, where $S' = \{s \in S_M \mid \mathrm{goal}_G(s) = \mathrm{true}\}$.*

For the sake of brevity, in the following, $\omega^*_M(s)$ abbreviates $\omega^*_M(s, S')$ with $S' = \{s \in S_M \mid goal_G(s) = true\}$, when $goal_G$ is clear from the context.

5 The Analysis of Graph Transition Systems

The two problems defined in the previous section can be solved with traditional graph exploration and shortest-path algorithms[3]. For the reachability problem, for instance, one can use, amongst others, depth-first search, hill climbing, best-first search, Dijkstra's algorithm (and its simplest version breadth-first search) or A*. For the optimality problem, only the last two are suited.

Recall that Dijkstra's algorithm [6] maintains a set of nodes as search horizon and iteratively explores the (currently optimal) node in the horizon. A* [18] basically improves Dijkstra's algorithm by selecting for exploration, the most promising node, i.e., it takes into account both the weight of the current optimal path to a node and an estimate on the distance to the set of goal nodes. Contrarily, best-first search takes into account the heuristic only. For a deeper treatment of both algorithms, we refer to [6, 18, 26].

Dijkstra's algorithm and A* are traditionally defined over a simple instance of our cost algebra A, namely cost algebra $\langle \mathbb{R}^+ \cup \{+\infty\}, +, \leq, +\infty, 0\rangle$. Fortunately, the results that ensure the *admissibility* of Dijkstra's algorithm or A*, i.e., the fact that both algorithms correctly solve the optimality problem, have been generalized for the cost algebra [13]:

Proposition 1 (Dijkstra is admissible). *Dijkstra's algorithm solves the optimality problem.*

Definition 11 (admissibility and consistency). *Given a GTS $\langle M, g\rangle$ and a goal function* goal_G*, a heuristic $h : S_M \to A$ is* admissible*, if for all $s \in S_M$ we have $h(s) \preceq \omega^*_M(s)$, and have $h(s) = \mathbf{1}$ whenever* $\mathrm{goal}_G(s)$*;* consistent*, if for each $s \xrightarrow{t} s'$, we have $h(s) \preceq \omega(t) \times h(s')$.*

A consistent heuristic is admissible if for all s such that $goal_G(s)$, $h(s) = \mathbf{1}$, even for our cost algebra [13].

Proposition 2 (A* is admissible). *A* solves the optimality problem if combined with admissible heuristics.*

[3] We refer here to a slight modification of the original algorithms, consisting of terminating the algorithm when a goal state is reached and returning the corresponding path.

6 Heuristics for Graph Transition Systems

As mentioned in the previous section, heuristic search algorithms use heuristic functions to guide the state space exploration. Here, we propose various heuristics for the analysis of graph transition systems.

6.1 Items to remove and insert

Consider Figure 1 and suppose we want to estimate the number of transitions necessary to transform the leftmost graph to the rightmost one. We need to remove edges e_3 and e_4, and to add edges e_6 and e_7. Since, we know that each transition removes and adds at most one edge, we can conclude that at least two transitions are necessary. These reasonings can be generalized as explained below.

First, recall that partial graph morphisms are induced by system transitions. In the case of graph transformation systems, for instance, graph morphisms are induced by graph transformation rules, while in communication protocols by the operations of the processes. In most cases, such transitions are usually local and involve a few insertion/deletion/merging of items. We can thus determine, prior to the analysis, the number of items deleted and erased by graph morphisms.

Let n^i_m and n^d_m respectively be the maximum number of inserted and deleted nodes in any transition, and e^i_m and e^d_m respectively be the maximum number of inserted and deleted edges in any transition, where the merging of n items is interpreted as the deletion of $n-1$ items. Let G be the goal graph. In addition, let c_m be the least cost associated to transitions. On the other hand, for goals of type 2, only the number of items to add are relevant. When considering goals of type 3, we cannot rely on the identity of edges as in previous heuristics and have thus to base our heuristic on the number of items to be added or deleted. Finally, for type 4, only the number of items to be added is taken into account.

Definition 12. *Heuristics $h^1_n, h^2_n, h^3_n, h^4_n$ are defined as follows:*

$$h^1_n(s) = c_m^{\max\{\lfloor |V_G \setminus V_{g(s)}|/n^i_m\rfloor, \lfloor |E_G \setminus E_{g(s)}|/e^i_m\rfloor, \lfloor |V_{g(s)} \setminus V_G|/n^d_m\rfloor, \lfloor |E_{g(s)} \setminus E_G|/e^d_m\rfloor\}}$$
$$h^2_n(s) = c_m^{\max\{\lfloor |V_G \setminus V_{g(s)}|/n^i_m\rfloor, \lfloor |E_G \setminus E_{g(s)}|/e^i_m\rfloor\}}$$
$$h^3_n(s) = c_m^{\max\{(\lfloor |V_G|-|V_{g(s)}|)/n^i_m\rfloor, (\lfloor |E_G|-|E_{g(s)}|)/e^i_m\rfloor, (\lfloor |V_{g(s)}|-|V_G|)/n^d_m\rfloor, (\lfloor |E_{g(s)}|-|E_G|)/e^d_m\rfloor\}}$$
$$h^4_n(s) = c_m^{\max\{(\lfloor |V_G|-|V_{g(s)}|)/n^i_m\rfloor, (\lfloor |E_G|-|E_{g(s)}|)/e^i_m\rfloor\}}$$

Proposition 3 **(h_n consistency).** *Heuristic h^1_n (resp. h^2_n,h^3_n,h^4_n) is consistent and, for goals of type 1 (resp. 2,3,4), admissible.*

6.2 Isomorphism Heuristics

The main drawback of the previously presented heuristics for goals of type 4 is evident. If state graphs have more edges and nodes than the goal graph, the resulting heuristic is completely blind, i.e., it returns **1** for all states. Thus A* degenerates into Dijkstra and best-first into a random search. Thus, we propose functions inspired by heuristics to decide isomorphism or sub-graph isomorphism. For instance, if one has to decide whether two graphs are isomorphic one would check first whether the two graphs have the same number of items. If so, one could continue trying to match nodes with the same in- and out-degrees.

First, let $d_{in}(u)$ and $d_{out}(u)$ denote the in- and out-degree of a node u in a graph G, i.e., $d_{in}(u) = |\{e \in E_G \mid tgt(e) = u\}|$ and $d_{out}(u) = |\{e \in E_G \mid src(e) = u\}|$. Let further D_G be the set of pairs of in- and out-degrees of all nodes of G, i.e., $D_G = \bigcup_{u \in V_G} \langle in(u), out(u) \rangle$, and $\hat{D}_G$ be a vector with all elements of D_G ordered according to the first component of the tuples. Finally, let d_M denote the Manhattan distance between two vectors, i.e., $d_M(u, v) = \sum_i |u_i - v_i|$.

Definition 13. *Let G be the goal graph. We define h_c^4 as*

$$h_c^4(s) = c_m^{\sum_{i=0}^{\max\{|\hat{D}_G|, |\hat{D}_{g(s)}|\}} d_M(\hat{D}_G[i], \hat{D}_{g(s)}[i])},$$

where $\hat{D}_{G'}[i]$ is $\langle 0, 0 \rangle$ if $i \geq |\hat{D}_{G'}|$. In words, we compute for each graph G and $g(s)$ a node degree-ordered vector. Then we compute the Manhattan distances of elements in the same rank. Intuitively, we decide a match of nodes and establish how many in- and out-going edges have to be removed or inserted.

Note that if one graph has more nodes than the other, we consider that the graph with less nodes has extra nodes with no degree at all. If the goal type is 3, we can refine the heuristic by trying different matches of the two vectors, as formalized in the following heuristic:

Definition 14. *Let G be the goal graph. We define h_c^3 as*

$$h_c^3(s) = \begin{cases} h_c^4(s) & \text{if } |\hat{D}_G| \geq |\hat{D}_{g(s)}| \\ c_m^{\bigsqcup \bigcup_{j=0}^{|\hat{D}_G| - |\hat{D}_{g(s)}|} \{\sum_{i=0}^{|\hat{D}_G|} d_M(\hat{D}_G[i+j], \hat{D}_{g(s)}[i])\}} & \text{otherwise} \end{cases}$$

Obviously, none of the two heuristics presented in this section is consistent or admissible in general, and one could define other versions of the heuristics by changing some of the parameters used: the order criteria, the distance between vectors, etc. The idea of these heuristics is, indeed, to illustrate the wide variety of non-admissible heuristics one could define.

6.3 Formula-Based Heuristic

Based on the original formula-based heuristic [14] we define a heuristic that exploits the specification of goal states by graph formulae. The details on how to transform the goal function as a goal graph and a requirement on the injective morphism into a corresponding closed negation-free FO graph formula is out of the scope of the paper. We consider here a simple example for Property 3. The corresponding formula is $\exists e. src(e) = tgt(e)$. In words, there is an edge with the same source and target.

In addition to boolean connectives, FO ingredients include first-order node and edge quantifiers, and node and edge comparison. For a detailed description of the logic we refer to [8]. The idea of the formula-based heuristic is that each false predicate contributes to an increase to the value. In other words, FO formulae are interpreted over the domain of the cost algebra in the spirit of quantitative logics [20]. Thus, *true* is interpreted as **1**, *false* as c_m, disjunction as selection and conjunction as cumulation.

Definition 15. *Let G be a graph and f, g be closed negation-free FO formulae. The interpretation of FO formulae over the cost algebra is given by*

$$\begin{array}{rlrl}
[\![\mathrm{true}]\!]^G &= \boldsymbol{1} & [\![\mathrm{false}]\!]^G &= c_m \\
[\![f \vee g]\!]^G &= [\![f]\!]^G \sqcup [\![g]\!]^G & [\![f \wedge g]\!]^G &= [\![f]\!]^G \times [\![g]\!]^G \\
[\![\exists x.f]\!]^G &= \bigsqcup_{u \in V_G} [\![f\{^u/_x\}]\!]^G & [\![\forall x.f]\!]^G &= \prod_{u \in V_G} [\![f\{^u/_x\}]\!]^G \\
[\![\exists y.f]\!]^G &= \bigsqcup_{e \in E_G} [\![f\{^e/_y\}]\!]^G & [\![\forall y.f]\!]^G &= \prod_{e \in E_G} [\![f\{^e/_y\}]\!]^G \\
[\![u = u']\!]^G &= \boldsymbol{1} \textit{ if } u = u',\ c_m \textit{ otherwise} & [\![e = e']\!]^G &= \boldsymbol{1} \textit{ if } e = e',\ c_m \textit{ otherwise}
\end{array}$$

where $\prod$ denotes the iterated application of operator $\times$, x and y are node and edge variables, respectively, and u, u' and e, e' are node and edge constants.

Finally, we define the formula-based heuristic as the interpretation of the formula described over the cost algebra.

Definition 16. *Heuristic h_f is defined as $h_f(s) = [\![f]\!]^{g(s)}$.*

The formula-based heuristic is neither consistent nor admissible in general as one transition can change change the falsehood of more than one predicate.

6.4 Hamming Distance

The Hamming distance of two bit vectors is the number of vector indices on which the bits differ. As there are many different encodings of a graph, we choose a simple one based on the image of the state representation in the memory.

Definition 17. *If bin_G denotes the bit-vector representation of G, and if we interpret* false *as 0, and* true *as 1, we obtain*

$$h_h(s) = c_m^{||bin_G| - |bin_{g(s)}|| + \sum_{i=0}^{\min\{|bin_G|, |bin_{g(s)}|\}} bin_G[i] \iff bin_{g(s)}[i]}$$

As more than one bit can change within one transition (e.g. the last one before reaching the goal) heuristic h_h is neither admissible nor consistent.

6.5 Tool-specific Heuristics

Finally, one can profit from specific heuristics available in the concrete tool that performs the analysis. For example, if the system is implemented and analyzed with HSF-SPIN [14], one can benefit from heuristics like the FSM Distance, which takes into account the finite automata representation of processes. If the Java Pathfinder [17] is used structural heuristics based on coverage metrics and interleavings are available. If planning tools like MIPS [12] are used, one can apply variants of the *relaxed planning heuristic* [21].detail in Section 8.

7 Abstraction and Heuristic Search

Abstraction is one of the most important issues to cope with large and infinite state spaces and to reduce the exploration efforts. Abstracted systems should be significantly smaller than the original one and while preserve some properties of concrete systems. The study of abstraction formalisms for graph transition systems is, however, out of the scope of this paper. We refer to [1] for an example of such a formalism. We assume that abstractions are available, state the properties necessary for abstractions to preserve our two problems (reachability and optimization) and propose how to use abstraction to define informed heuristics.

The preservation of the reachability problem means that the existence of an initial goal path in the concrete system must entail the existence of a corresponding initial goal path in the abstract system. Note that this does not mean the existence of *spurious* initial goal paths in the abstract system, i.e., abstract paths that do no not correspond to any concrete path. Similarly, the preservation of the optimization problem means that the cost of the optimal initial goal path in the concrete system should be greater or equal to the cost of the optimal initial goal path in the abstract system.

Abstractions have been applied in combination with heuristic search both in model checking [15] and planning [10] approaches. The main idea is that the abstract system is explored in order to create a database that stores the exact distances from abstract states to the set of abstract goal states. The exact distance between abstract states is an admissible and consistent estimate of the distance between the corresponding concrete states. The distance database is thus used as heuristics for analyzing the concrete system.

We recall the notion of $\langle \alpha, \gamma \rangle$-simulations [24] typically used in model checking abstraction approaches.

Definition 18 (Galois connection). *Let S, S' be two set of states. A* Galois connection *from 2^S to $2^{S'}$ is a pair of monotonic functions $\langle \alpha, \gamma \rangle$, with $\alpha : 2^S \rightarrow 2^{S'}$ (abstraction) and $\gamma : 2^{S'} \rightarrow 2^S$ (concretization) such that $\mathrm{id}_{2^S} \subseteq \gamma \circ \alpha$ and $\alpha \circ \gamma \subseteq \mathrm{id}_{2^{S'}}$.*

Definition 19 (simulation). *Let $\langle M, g \rangle$ and $\langle M', g' \rangle$ be two GTSs and $\langle \alpha, \gamma \rangle$ be a Galois connection from 2^{S_M} to $2^{S_{M'}}$. We say that $\langle M', g' \rangle$ $\langle \alpha, \gamma \rangle$-simulates $\langle M, g \rangle$, written $\langle M, g \rangle \sqsubseteq_{\langle \alpha, \gamma \rangle} \langle M', g' \rangle$, if $\alpha \circ \mathrm{pre} \circ \gamma \subseteq \mathrm{pre}'$, where $\mathrm{pre} : S_M \rightarrow S_M$ is defined by $\mathrm{pre}(S) = \{s \in S \mid s \xrightarrow{t} s'\}$ and pre' is defined similarly for $\langle M', g' \rangle$.*

We say that a simulation $\langle M, g \rangle \sqsubseteq_{\langle \alpha, \gamma \rangle} \langle M', g' \rangle$ *preserves* a goal function $goal_G$ whenever $s' \in \alpha(s')$ implies $goal_G(s) \Rightarrow goal_G(s')$, and we call it *cost consistent* if for any transition $s_1 \xrightarrow{t} s_2$ in M there is a transition $s_1' \xrightarrow{t'} s_2'$ with $s_1' \in \alpha(s_1)$, $s_2' \in \alpha(s_2)$ and $\omega(t') \preceq \omega(t)$.

Proposition 4 (reachability preservation). *Let $\langle M, g \rangle$, $\langle M', g' \rangle$ be two GTSs such that $\langle M, g \rangle \sqsubseteq_{\langle \alpha, \gamma \rangle} \langle M', g' \rangle$ and goal_G is preserved. Then, there is a solution to the reachability problem in $\langle M', g' \rangle$ if there is a solution to the reachability problem in $\langle M, g \rangle$.*

As a consequence, if there is no solution to the reachability problem in the abstract graph transition system, there is no solution in the concrete system. Recall that the contrary is not true: there might be initial goal paths in the abstract system, but not in the concrete one. Such *spurious* solutions are usually eliminated by refining the abstractions [19]. Now we state that the optimality problem is preserved for cost consistent simulations.

Proposition 5 (optimality preservation). *Let $\langle M, g\rangle$, $\langle M', g'\rangle$ be two GTSs such that $\langle M, g\rangle \sqsubseteq_{\langle\alpha,\gamma\rangle} \langle M', g'\rangle$ is cost consistent and* goal_G *is preserved. Then $\omega^*_{M'}(s_0^{M'}) \preceq \omega^*_M(s_0^M)$.*

We now describe how to use abstraction to define informed heuristics.

Definition 20 (abstract database heuristic). *Let $\langle M, g\rangle$, $\langle M', g'\rangle$ be two GTSs such that $\langle M, g\rangle \sqsubseteq_{\langle\alpha,\gamma\rangle} \langle M', g'\rangle$ is cost consistent and* goal_G *is preserved. Heuristic h_a is defined as $h_a(s) = \omega^*_{M'}(s')$, for any $s' \in S_{M'}$ such that $s' \in \alpha(s)$.*

Proposition 6. *Heuristic h_a is consistent and admissible.*

When different abstractions are available, we can combine the different databases in various ways to obtain better heuristics. The first way is to trivially select the best value delivered by two heuristic databases, which trivially results in a consistent and admissible heuristic.

Definition 21 (selective combination). *Given two different abstraction database heuristics h_a and $h_{a'}$ we define $h_{a\sqcup a'}$ as $h_{a\sqcup a'}(s) = h_a(s) \sqcup h_{a'}(s)$.*

In some cases, however, it is possible to take their cumulative values using $\times$, which provides a much better guidance for the search process. The corresponding abstraction databases are called *disjoint*. Intuitively the idea is that each (non self-)transition in the concrete system either has a corresponding (non self-)transition in one of the abstracted systems but not in both.

Definition 22 (disjoint abstractions). *Let $\langle M, g\rangle$ be a GTS and $\langle M', g'\rangle$, $\langle M'', g''\rangle$ be two abstracted GTS such that $\langle M, g\rangle \sqsubseteq_{\langle\alpha,\gamma\rangle} \langle M', g'\rangle$, $\langle M, g\rangle \sqsubseteq_{\langle\alpha',\gamma'\rangle} \langle M'', g''\rangle$ are cost consistent and* goal_G *is preserved by both simulations. We say that $\langle M', g'\rangle$, $\langle M'', g''\rangle$ are* disjoint abstractions *whenever for any transition $s_1 \xrightarrow{t} s_2$ in M such that $s_1 \neq s_2$ either there is a transition $s'_1 \xrightarrow{t} s'_2$ with $s'_1 \neq s'_2$ and $s'_1 \in \alpha(s_1)$, $s'_2 \in \alpha(s_2)$, or there is a transition $s''_1 \xrightarrow{t} s''_2$ with $s''_1 \neq s''_2$ and $s''_1 \in \alpha'(s_1)$, $s''_2 \in \alpha'(s_2)$.*

Figure 3 depicts a concrete transition system (left) with three abstractions (given by node mergings). The center-left and center-right abstractions are mutually disjoint. However any of these together with the rightmost abstraction is not disjoint. For instance, the concrete transition from s_0 to s_3 in the leftmost graph has a corresponding abstract (non self-)transition in the center-left abstraction and in the rightmost one. As a result the distance from s_0 to s_3 would be estimated as 3 which is clearly not a lower bound.

$$\begin{tikzcd}
s_3 & s_2 \arrow[l] \\
s_1 \arrow[u] & s_0 \arrow[l] \arrow[u]
\end{tikzcd}
\qquad
\begin{tikzcd}
s_2, s_3 \arrow[loop above] & s_1, s_0 \arrow[l] \arrow[loop above]
\end{tikzcd}
\qquad
\begin{tikzcd}
s_1, s_3 \arrow[loop above] & s_2, s_0 \arrow[l] \arrow[loop above]
\end{tikzcd}
\qquad
\begin{tikzcd}
s_3 & s_1, s_2 \arrow[l] & s_0 \arrow[l]
\end{tikzcd}$$

Fig. 3. A transition system (leftmost) with three different abstractions.

Definition 23 (cumulative combination). *Given two different abstraction database heuristics h_a and $h_{a'}$, we define $h_{a \sqcup a'}$ as $h_{a \sqcup a'}(s) = h_a(s) \times h_{a'}(s)$.*

Proposition 7. *Let $\langle M, g \rangle$ be a GTS and $\langle M', g' \rangle$, $\langle M'', g'' \rangle$ be two disjoint abstracted GTS. Let $\langle M, g \rangle$ be a GTS and $\langle M', g' \rangle$, $\langle M'', g'' \rangle$ be two abstracted GTS such that $\langle M, g \rangle \sqsubseteq_{\langle \alpha, \gamma \rangle} \langle M', g' \rangle$, $\langle M, g \rangle \sqsubseteq_{\langle \alpha', \gamma' \rangle} \langle M'', g'' \rangle$ are cost consistent and* goal_G *is preserved by both simulations. Let further h_a and $h_{a'}$ be the database heuristics constructed from $\langle M', g' \rangle$ and $\langle M'', g'' \rangle$, respectively. Then $h_{a \sqcup a'}$ is consistent and admissible.*

8 Experimental Results

We validate our approach by presenting experimental results obtained with HSF-SPIN [14], a heuristic model checker compatible with the successful model checker SPIN [22]. The analysis we perform regards Property 2, i.e., *Can a certain node v_i be a terminal and no other requests are queued over the network?*. We have implemented the Arrow Distributed Directory Protocol in Promela, the specification language of both SPIN and HSF-SPIN. The implemented model allows for an easy definition of the minimal spanning tree underlying the protocol. A node is modeled as a non-deterministic process that can request an object, accept the request as a non-terminal node, accept the request as a terminal node, send the object if it has finished working on it, or receive the object sent directly over the network. We choose three different topologies: *star*, where all nodes are connected to one common node, *chain*, nodes forming a connected chain, and *tree*, where nodes are arranged in the form of a binary tree. Each instance consists of 10 nodes. In all our experiments, we set a memory bound of 512 MB.

The results in Table 1 correspond to the first phase of the goal-finding process, namely when one is interested in finding a goal state as quick as possible. The table shows the number of expanded nodes and solution length for Dijkstra's algorithm (DJK), depth-first search (DFS) and best-first search with heuristics h_n^1 (BF+h_n^1), h_h (BF+h_h) and h_f (BF+h_f).

As expected, Dijkstra's algorithm offers the optimal path to the desired state graph though requiring the greatest number of state expansions. Best-first offers the best performance in terms of node expansions with heuristics h_n^1 and h_f. It is worth mentioning that in this particular example h_f amounts to $(h_n^1)^5$. When applying the Hamming distance, the number of expanded nodes increases, still the solution length decreases.

star	DJK	DFS	BF+h_n^1	BF+h_h	BF+h_f
expanded nodes	38,701	6,253	30	6,334	30
solution cost	20	134	58	32	58
chain	DJK	DFS	BF+h_n^1	BF+h_h	BF+h_f
expanded nodes	413,466	78,112	38	1,49	38
solution cost	28	118	74	74	74
tree	DJK	DFS	BF+h_n^1	BF+h_h	BF+h_f
expanded nodes	126,579	24,875	34	24,727	34
solution cost	24	126	66	44	66

Table 1. Reachability experiments in the arrow distributed directory protocol.

star	DJK	A*+$H_h(s_d)$	A*+$H_h(s_b)$	A*+h_n^1	A*+h_h	A*+h_f
expanded nodes	38,701	o.m.	1,255	13,447	117	206
solution cost	20	o.m.	20	20	20	20
chain	DJK	A*+$H_h(s_d)$	A*+$H_h(s_b)$	A*+h_n^1	A*+h_h	A*+h_f
expanded nodes	413,466	26,622	1,245	106,629	1,620	198
solution cost	28	42	28	28	28	28
tree	DJK	A*+$H_h(s_d)$	A*+$H_h(s_b)$	A*+h_n^1	A*+h_h	A*+h_f
expanded nodes	126,579	o.m	1,481	33,720	6,197	224
solution cost	24	o.m.	24	24	24	24

Table 2. Optimality experiments in the arrow distributed directory protocol.

Table 2 regards the second phase of the bug-finding process, namely when one is interested in finding optimal paths to a given goal state. The table shows the number of expanded nodes and solution length for Dijkstra's algorithm (DJK) and A* with heuristics h_n^1 (A*+h_n^1), h_h (A*+h_h) and h_f (A*+h_f). In addition, we used the Hamming distance applied to the whole state vector representation (A*+H_h), where s_b and s_d indicate that the goal state used for the heuristic was the one obtained with breadth- and depth-first search, respectively.

The first thing to observe is that h_n^1, being admissible, always delivers optimal paths and requires less search effort than Dijkstra's algorithm. The performance of the Hamming distance is not regular. Version h_h that takes into account the graph representation and $H_h(s_b)$ based on the whole bit-vector of the state obtained with Dijkstra's algorithm outperform heuristic h_n^1. On the other hand, the Hamming distance that takes into account the bit-vector representation of the state obtained with the depth-first search exploration shows poor performances by delivering non-optimal counterexamples and running out of memory. The reason is that the state vectors corresponding to states s_b and s_d are very different (though both representing the goal state). The bits representing data, not involving the goal graph items, result in rich information in one case and fuzzy in the other. Heuristic h_f performs the best, both in terms of the expanded nodes and the path length.

9 Conclusion

We have presented an abstract approach for the analysis of graph transitions systems, which are traditional transition systems where states and transitions respectively represent graphs and partial graph morphisms. It is a useful formalism to represent the state space of systems involving graphs, like communication protocols, graph transformations, and visually described systems.

The analysis of such systems is reduced to exploration problems consisting of finding certain states reachable from the initial one. We analyze two problems: finding just one path and finding the optimal one, according to a certain notion of optimality. As algorithms, we propose the use of heuristic search. They use heuristic functions that lead the exploration to the set of goal states. We have proposed different such functions proving some of their properties. In addition, we have proposed the use of abstraction-based heuristics which exploit abstraction techniques in order to obtain informed heuristics.

We have illustrated our approach with a scenario in which one is interested in analyzing structural properties of communication protocols. As a concrete example we used the *arrow distributed directory protocol* [9], which ensures exclusive access to a mobile service in a distributed system. We implemented our approach in the heuristic model checker HSF-SPIN, an extension of the well-known model checker SPIN and presented promising preliminary experiments. In future work, we plan to realize a richer empirical evaluation of our approach, focusing on abstraction database heuristics and possibly profiting from existing approaches for the abstraction of graph transformation systems[1, 28]

References

1. P. Baldan, A. Corradini, B. König, and B. König. Verifying a behavioural logic for graph transformation systems. In *CoMeta'03*, ENTCS, 2004.
2. B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.
3. L. Caires and L. Cardelli. A spatial logic for concurrency (part I). *Inf. Comput.*, 186(2):194–235, 2003.
4. L. Cardelli, P. Gardner, and G. Ghelli. A spatial logic for querying graphs. In *ICALP'2002*, volume 2380 of *Lecture Notes in Computer Science*, pages 597–610. Springer, 2002.
5. E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* The MIT Press, 1999.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.
7. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. *Algebraic approaches to graph transformation*, volume 1, chapter Basic concepts and double push-out approach. World Scientific, 1997.

8. B. Courcelle. *Handbook of graph grammars and computing by graph transformations*, volume 1 : Foundations, chapter 5: The expression of graph properties and graph transformations in monadic second-order logic, pages 313–400. World Scientific, 1997.
9. M. J. Demmer and M. Herlihy. The arrow distributed directory protocol. In *International Symposium on Distributed Computing (DISC 98)*, pages 119 –133, 1998.
10. S. Edelkamp. Planning with pattern databases. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science. Springer, 2001. 13-24.
11. S. Edelkamp. Promela planning. In *Model Checking Software (SPIN)*, pages 197–212, 2003.
12. S. Edelkamp. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Research (JAIR)*, 20:195–238, 2003.
13. S. Edelkamp, S. Jabbar, and A. Lluch Lafuente. Cost-algebraic heuristic search. In *National Conference on Artificial Intelligence (AAAI)*, pages 1362–1367, 2005.
14. S. Edelkamp, S. Leue, and A. Lluch Lafuente. Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools for Technology Transfer (STTT)*, 5(2-3):247–267, 2003.
15. S. Edelkamp and A. Lluch Lafuente. Abstraction databases in theory and model checking practice. In *ICAPS Workshop on Connecting Planning Theory with Practice*, 2004.
16. F. Gadducci and A. Lluch Lafuente. Graphical verification of a spatial logic for the pi-calculus. In *Proceedings of the 1st Workshop on Graph Transformation for Verification and Concurrency*. Elsevier Electronic Notes in Theoretical Computer Science, 2005. to appear.
17. A. Groce and W. Visser. Model checking Java programs using structural heuristics. In *International Symposium on Software Testing and Analysis (ISSTA)*. ACM Press, 2002.
18. P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for heuristic determination of minimum path cost. *IEEE Transactions on on Systems Science and Cybernetics*, 4:100–107, 1968.
19. T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software verification with Blast. In *Proceedings of the 10th SPIN Workshop on Model Checking Software (SPIN)*, volume 2648 of *Lecture Notes in Computer Science*, pages 235–239, 2003.
20. D. Hirsch, A. Lluch Lafuente, and E. Tuosto. A logic for application level QoS. *Theoretical Computer Science, Special Issue on Quantitative Aspects of Programming Languages*, 2005. To Appear.
21. J. Hoffmann and B. Nebel. Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.
22. G. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.
23. R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.
24. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–35, 1995.
25. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
26. J. Pearl. *Heuristics*. Addison-Wesley, 1985.
27. A. Rensink. Towards model checking graph grammars. In *3rd Workshop on Automated Verification of Critical Systems*, Tech. Report DSSE-TR-2003, pages 150–160, 2003.

28. A. Rensink and D. Distefano. Abstract graph transformation. In *International Workshop on Software Verification and Validation (SVV)*, Electronic Notes in Theoretical Computer Science, 2005. To appear. Technical report version: CTIT TR–CTIT–05–04, University of Twente.
29. G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformations.* World Scientific, 1997.
30. J. Sobrinho. Algebra and algorithms for QoS path computation and hop-by-hop routing in the internet. *IEEE/ACM Trans. Netw.*, 10(4):541–550, 2002.
31. D. Varrò. Automated formal verification of visual modeling languages by model checking. *Journal on Software and Systems Modeling*, 2003.
32. I. Wegener. *Komplexitätstheorie.* Springer, 2003. (in German).

A Proofs

Proposition 3. (h_n consistency) *Heuristic h^1_n (resp. h^2_n,h^3_n,h^4_n) is consistent and, for goals of type 1 (resp. 2,3,4), admissible.*

Proof. We show the proposition for h^1_n since the rest of the proofs are very similar. We have to show that $h^1_n(s) \preceq \omega(t) \times h^1_n(s')$ for every transition $s \xrightarrow{t} s'$:

$$\begin{aligned}
h^1_n(s) &\overset{(1)}{=} c_m^{\max\{|V_G \setminus V_{g(s)}|/n^i_m, |E_G \setminus E_{g(s)}|/e^i_m, |V_{g(s)} \setminus V_G|/n^d_m, |E_{g(s)} \setminus E_G|/e^d_m\}} \\
&\overset{(2)}{\preceq} c_m^{\max\{(|V_G \setminus V_{g(s')}| + |V_{g(s')} \setminus V_{g(s)}|)/n^i_m, (|E_G \setminus E_{g(s')}| + |E_{g(s')} \setminus E_{g(s)}|)/e^i_m, (|V_{g(s')} \setminus V_G| + |V_{g(s)} \setminus V_{g(s')}|)/n^d_m, (|E_{g(s')} \setminus E_G| + |E_{g(s)} \setminus E_{g(s')}|)/e^d_m\}} \\
&\overset{(3)}{\preceq} c_m^{\max\{|V_G \setminus V_{g(s')}|/n^i_m, |E_G \setminus E_{g(s')}|/e^i_m, |V_{g(s')} \setminus V_G|/n^d_m, |E_{g(s')} \setminus E_G|/e^d_m\}} \times \\
&\qquad c_m^{\max\{|V_{g(s')} \setminus V_{g(s)}|/n^i_m, |E_{g(s')} \setminus E_{g(s)}|/e^i_m, |V_{g(s)} \setminus V_{g(s')}|/n^d_m, |E_{g(s)} \setminus E_{g(s')}|/e^d_m\}} \\
&\overset{(4)}{=} h^1_n(s') \times c_m^{\max\{|V_{g(s')} \setminus V_{g(s)}|/n^i_m, |E_{g(s')} \setminus E_{g(s)}|/e^i_m, |V_{g(s)} \setminus V_{g(s')}|/n^d_m, |E_{g(s)} \setminus E_{g(s')}|/e^d_m\}} \\
&\overset{(5)}{\preceq} h^1_n(s') \times \omega(t)
\end{aligned}$$

We have (1) by definition, (2) by isotonicity of $\times$, $|V_G \setminus V_{g(s)}| \leq |V_G \setminus V_{g(s')}| + |V_{g(s')} \setminus V_{g(s)}|$ and analogous, (3) because $\max\{a+b, c+d\} \leq \max\{a, c\} + \max\{b + d\}$ and isotonicity of $\times$, and (4), (5) by definition.

Next, for goals of type 1 it is easy to see that $h^1_n(s) = \mathbf{1}$ if $goal_G(s) = true$. Together with consistency this implies that h^1_n is admissible for goals of type 1.

Proposition 4. (reachability preservation) *Let $\langle M, g\rangle$, $\langle M', g'\rangle$ be two GTSs such that $\langle M, g\rangle \sqsubseteq_{\langle\alpha,\gamma\rangle} \langle M', g'\rangle$ and* goal_G *is preserved. Then, there is a solution to the reachability problem in $\langle M', g'\rangle$ if there is a solution to the reachability problem in $\langle M, g\rangle$.*

Proof. As usual, a simulation relation preserves paths, i.e., if there is a path in M from a state s_1 to a state s_2, then there is a path in M' from s'_1 to a state s'_2 such that $s'_1 \in \alpha(s_1)$ and $s'_2 \in \alpha(s_2)$. Hence, for an initial goal path in M from

s_0^M to a state s we have a corresponding path in M' from s_0^M to a state s'. The assumption on $\langle\alpha, \beta\rangle$ implies $goal_G(s') = true$.

Proposition 5. (optimality preservation) *Let $\langle M, g\rangle$, $\langle M', g'\rangle$ be two GTSs such that $\langle M, g\rangle \sqsubseteq_{\langle\alpha,\gamma\rangle} \langle M', g'\rangle$ is cost consistent and* goal_G *is preserved. Then $\omega^*_{M'}(s_0^{M'}) \preceq \omega^*_M(s_0^M)$.*

Proof. Let $p = s_0^M \overset{s_1}{\rightarrow} \dots s_k$ be an optimal initial goal path in M starting at a given node s that is associated with G'. Then $\alpha(p) = \alpha(s_0) \overset{\alpha(t_0)}{\rightarrow} \alpha(s_1) \dots \phi(s_k)$ is a solution path in the abstract model M', which cost is necessarily lower than the cost of p. In addition, the cost of the optimal solution path in the abstract model can only be better.

Proposition 6. (abstract database heuristic) *Heuristic h_a is consistent and admissible.*

Proposition 7. *Let $\langle M, g\rangle$ be a GTS and $\langle M', g'\rangle$, $\langle M'', g''\rangle$ be two disjoint abstracted GTS. Let $\langle M, g\rangle$ be a GTS and $\langle M', g'\rangle$, $\langle M'', g''\rangle$ be two abstracted GTS such that $\langle M, g\rangle \sqsubseteq_{\langle\alpha,\gamma\rangle} \langle M', g'\rangle$, $\langle M, g\rangle \sqsubseteq_{\langle\alpha',\gamma'\rangle} \langle M'', g''\rangle$ are cost consistent and* goal_G *is preserved by both simulations. Let further h_a and $h_{a'}$ be the database heuristics constructed from $\langle M', g'\rangle$ and $\langle M'', g''\rangle$, respectively. Then $h_{a\sqcup a'}$ is consistent and admissible.*

Proof. To show consistency, we have to proof $h_{a\sqcup a'}(s_1) \preceq_{a\sqcup a'} (s_2) \times \omega(t)$. By definition $h_{a\sqcup a'}(s_1) = h_a(s_1) \times h_{a'} \times \omega(t)$, which is trivially smaller or equal than $\omega_{M'}(t') \times \omega^*_{M'}(s_2') \times \omega_{M''}(t'') \times \omega^*_{M''}(s_2')$ for any $s_1' \overset{t'}{\rightarrow} s_2'$, $s_1'' \overset{t''}{\rightarrow} s_2''$ with $s_1' \in \alpha(s_1)$, $s_2' \in \alpha(s_2)$, $s_1'' \in \alpha'(s_1)$ and $s_2''\alpha'(s_2)$. Since both abstraction are disjoint we know that $s_1' = s_2'$ or $s_1'' = s_2''$. Hence, it is easy to see that $\omega_{M'}(t') \times \omega^*_{M'}(s_2') \times \omega_{M''}(t'') \times \omega^*_{M''}(s_2') \preceq h_a(s_2') \times \omega(t)$.