Real-Time Model Checking on Secondary Storage

Stefan Edelkamp and Shahid Jabbar

University of Dortmund
Otto-Hahn Straße 14
Germany
{stefan.edelkamp,shahid.jabbar}@cs.uni-dortmund.de [⋆]

Abstract. In this paper, we consider disk based exploration in priced timed automata for resource-optimal scheduling. State spaces for large problems can easily go beyond the main memory capacity. We propose the use of hard disk to store the generated state space induced by priced timed automata. We contribute three algorithms: External Breadth First Search for reachability analysis in ordinary timed automata, External Breadth First Branch-and-Bound for cost-optimal reachability analysis in priced timed automata, and Iterative Broadening External Breadth First Branch-and-Bound for a partial exploration in priced timed automata. The third algorithm achieves its completeness by trying to find an upper bound on the optimal solution in an incomplete search tree. Iteratively, the upper bound is made tighter and the coverage of the search space is widened. We present correctness and completeness proofs for the suggested algorithms along with experimental results on different instances of aircraft landing scheduling to validate the practicality of our approach.

1 Introduction

Real-time model checking with timed automata [2] is an important decidable subfield of the analysis of hybrid automata [11] with a number of industrial applications. UPPAAL [21] is one very successful verification tool based on timed automata. It can be used for modeling, simulation and validation of real-time systems. It deals with non-deterministic processes with finite control structure, channel or shared variable communication, and real-valued clocks. UPPAAL CORA [20] is the extension of UPPAAL designed for efficient cost-optimal reachability analysis in priced timed automata. UPPAAL CORA is also competitive in resource-optimal scheduling [23].

The main limitation to the exploration of real-time systems are bounded main memory resources. Relying on virtual memory slows down the exploration due to excessive page faults. External algorithms [24] exploit harddisk space and organize the access to secondary memory. Originally designed for explicit graphs, external search algorithms have shown considerable performances in the large-scale breadth-first and guided exploration of single-agent games [16, 9] and in the analysis of model checking problems [13, 14, 18]. While [14] provides a distributed implementation of [13] for model checking safety properties, a recent extension [8] extends the approach to general LTL

[⋆] The authors are supported by DFG under projects *Directed Model Checking* and *Heuristic Search*.

properties. The approaches in [8, 13, 14] have been implemented on top of Spin model checker and have succeeded in exploring state spaces as large as 3 Terabytes. In [25] the model checker Murϕ has been extended to use hard disk to store intermediate states.

In this paper, we extend external search algorithms for exploration in unweighted and weighted real-time models. The challenge is to I/O efficiently deal with the external representation and elimination of redundant states. We propose three algorithms: External Breadth First Search for reachability analysis in ordinary timed automata, External Breadth First Branch-and-Bound for cost-optimal reachability analysis in priced timed automata, and Iterative Broadening External Breadth First Branch-and-Bound for a partial exploration in priced timed automata. The proposed algorithms provide a controlled and guided exploration of the state space.

The paper is structured as follows. First, we review real-time model checking with priced timed automata. Then, we consider external exploration and introduce delayed duplication detection in breadth-first search. Next, we present external search in real-time domains. An introduction to priced timed automata is presented next. Since in the priced timed automata, we are interested in a cost optimal solution, we combine external search with branch-and-bound. Later, we present an iterative broadening variant of the algorithm that tries to find a good upper bound by searching in only a fragment of the state space. We have implemented our approach in UPPAAL CORA. Results for various problems of aircraft landing scheduling are presented.

In this text we consider real-time model checking with timed automata, for which the reachability problem is decidable but PSPACE-hard [2]. We furthermore restrict overselves to the cost optimization variant of reachability analysis for linearly priced timed automata. For extending these explorations to real-time model checking with respect to temporal properties we refer the reader to [6]. Moreover,

2 Timed Automata

Timed Automata can be viewed as an extension of classical finite automata with clocks and constraints defined on these clocks. These constraints, when corresponding to states are called *invariants*, and restrict the time allowed to stay at the state. When corresponding to transitions these constraints are called *guards*, and restrict the use of the transition. The clocks C are real-valued variables and are used to measure durations. The values of all the clocks in the system are denoted as a vector, also called as clock valuation function $v : C \to I\!\!R^+$. The constraints are defined over clocks and can be generated by the following grammar: for $x, y \in C$, a constraint α is defined as,

$$\alpha ::= x \prec d \mid x - y \prec d \mid \neg\alpha \mid (\alpha \wedge \alpha),$$

where $d \in Z\!\!\!Z$ and $\prec \in \{<, \leq\}$. These constraints yield two different kinds of transitions. The first one (*delay* transition) is to wait for some duration in the current state s - provided the *invariant*(s) holds. This lets only the clock variables increase. The other operation (*edge* transition) resets some clock variables while taking the transition t. The operation is possible given that the *guard*(t) holds. We allow an edge transition to be taken without an increase in the clock variables, i.e., in time 0. *Trajectories* are alternating sequences of states and transitions and define a path within the automata. The reachability task is to determine, if the goal in form of partial assignment to the ordinary and clock variables can be reached or not. The optimal reachability problem is to find a trajectory that minimizes the overall path length.

For a reachability analysis on timed automata, one faces the problem of an infinite-state space. This infiniteness is due to the fact that the clocks are real-valued and, hence, an exhaustive state space exploration can yield to infinite branches. This problem was solved with the introduction of a partitioning scheme based on regions [2]. A region automata creates finitely many partitions of the infinite state space based on the equivalent classes of the clock valuations. In model checking tools like Uppaal, though, a coarser representation called as *zone* [2] is used. Formally, a *zone* Z over a set of clocks C is a finite conjunction of simple difference constraints of the form $x - y \leq d$ or $x - y < d$, with $x, y \in C$ and integer d[1]. The semantics for delay and edge transitions in a timed automata are based on some basic operations. We restrict to changes in clock variables. For a clock vector u and a zone Z we write $u \in Z$ if u satisfies the constraints in Z. The two main operations on (clock) zones are clock *reset* $\{x\}Z = \{u[0/x] \mid u \in Z\}$ that resets all the clocks x, *delay* or *future* (d time units) $Z^{\uparrow} = \{u + d \mid u \in Z\}$. The reachability problem in timed automata can then be reduced to the reachability analysis in *zone automata*. In a zone automata, each state is basically a *symbolic state* corresponding to one or many states in the original Timed Automata. The new state is represented as a tuple (l, Z), with l being the discrete part containing the local state of the automata, and Z is the convex $|C|$-dimensional hypersurface in Euclidean space. Semantically, (l, Z) now represents the set of all states (l, u) with $u \in Z$. Let $\mathcal{B}(C)$ denotes the set of constraints defined on clocks C and $\mathcal{P}(C)$ the power set of C. Formally, a Timed automata can be defined as follows:

Definition 1 (Timed Automata). *A timed automata is a tuple* $A = (\mathcal{S}, l_0, \mathcal{R}, \mathrm{Inv}, \mathcal{T})$*, where* $\mathcal{S}$ *is the set of states,* (l_0, Z_0) *is the initial state with an empty zone,* $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{B}(C) \times \mathcal{P}(C) \times \mathcal{S}$ *is the transition relation making states to their successors, given the constraints on the edge are satisfied,* $\mathrm{Inv} : \mathcal{S} \rightarrow \mathcal{B}(\mathcal{C})$ *assigns invariants to the states, and* $\mathcal{T}$ *is the set of final states.*

3 External Breadth First Search

Most modern operating systems hide secondary memory accesses from the programmer, but offer one consistent address space of *virtual memory* that can be larger than the internal memory. When the program is executed, virtual addresses are translated into physical addresses. Only those portions of the program currently needed for the execution are copied into main memory. Caching and pre-fetching heuristics have been developed in order to reduce the number of page faults (the referenced page does not reside in the cache and has to be loaded from the hard disk). However, these methods are general-purpose and can not always take full advantage of the locality inherent in algorithms. Algorithms that explicitly manage the memory hierarchy can lead to substantial speedups, since they are more informed to predict and adjust future memory access.

[1] Unary constraints $x \leq d$ or $x < d$ are rewritten as $x - x_0 \leq d$ and $x - x_0 < d$ for some start time clock variable x_0, $x - y \geq d$ as $y - x \leq -d$ and $x = y$ as $x - y \leq 0$ and $y - x \leq 0$.

The standard model for comparing the performance of external algorithms consists of a single processor, a small internal memory that can hold up to M data items, and an unlimited secondary memory. The size of the input problem (in terms of the number of records) is abbreviated by N. Moreover, the *block size* B governs the bandwidth of memory transfers[2]. Typically $M = \sqrt{B}$. It is usually assumed that at the beginning of the algorithm, the input data is stored in contiguous block on external memory, and the same must hold for the output. Only the number of block reads and writes are counted, computations in internal memory do not incur any cost. The single disk model for external algorithms has been invented by [1]. It is convenient to express the complexity of external-memory algorithms using a number of frequently occurring primitive operations:

1. *scan*(N) with an I/O complexity of $\Theta(\frac{N}{B})$ that can be achieved through trivial sequential access.
2. *sort*(N) with an I/O complexity of $\Theta(\frac{N}{B}\log_{M/B}\frac{N}{B})$ that can be achieved through *External Merge* or *Distribution Sort*

Finite State Systems One of the first efforts towards a search algorithm that works on external memory is due to Munagala and Ranade [22]. The authors presented an external memory Breadth First Search(BFS) algorithm for explicit graphs, i.e., the graphs that are completely available beforehand in the form of adjacency lists. For example, a road network. Later the algorithm has been adapted for the implicit graphs that are generated on-the-fly from an initial state and a set of rules/transitions, and has been called *delayed duplicate detection* for *frontier search*. Both of the these algorithms assume an unweighted and undirected graph and work on a similar principle. Let *Succ* be the successor generation function. The algorithms maintain BFS layers on disk[3]. Let *Open*(j) represent the set of states at layer j. Layer *Open*$(j-1)$ is scanned and the set of successors is put into a buffer of size close to the main memory capacity. If the buffer becomes full, internal sorting followed by a scanning generates a sorted duplicate-free state sequence in the buffer that is flushed to disk. This results in a file with states belonging to depth j stored in the form of sorted buffers. To remove the duplicates, *external sorting* is applied to unify the buffers into one sorted file. Due to sorting, all duplicates will come close to each other and a simple scan is enough to generate a duplicate free file. One also has to eliminate/subtract previous layers from *Open*(j) to avoid re-expansions. In [22], the authors argue that for undirected graphs, subtracting two previous layers is enough to guarantee that no state is expanded twice.

The process is repeated until *Open*$(j-1)$ becomes empty, or the goal has been found. Delayed duplicate detection applies $O(\mathit{sort}(|\mathit{Succ}(\mathit{Open}(j-1))|)+\mathit{scan}(|\mathit{Open}(j-$

[2] On the hardware level the block size B is fixed by the computer architecture. From the application program point of view it is possible to vary B according to the given resources. If only a constant number c of internal buffers are required, the block size can be scaled to $M = cB$.

[3] As BFS traverses the graph in layers, only two active files are needed, one for reading the expanded states and one for writing the generated states. To I/O optimally cope with sparse graphs, the BFS layers can be maintained in one large file together with file pointers locating their offsets and with two internal buffers for reading and writing. With respect to the previous footnote this implies that $M = 2B$.

$1)|+|Open(j-2)|))$ I/Os. Since each edge contributes to one state, $\sum_j |Succ(Open(j))| = O(|\mathcal{R}|)$ and $\sum_j |Open(j)| = O(|\mathcal{S}|)$. This gives a total I/O complexity of $O(sort(|\mathcal{R}|) + scan(|\mathcal{S}|))$ I/Os, which – assuming delayed duplicate detection on general state vectors is needed – proves to be optimal [3].

The algorithm shares similarities with internal *Frontier Search* [15, 17] that was used for solving multiple sequence alignment problems, an idea that goes back to Hirschberg [12]. The sorting complexity can be improved in practice by using a hash-based delayed duplicate detection scheme. Frontier search has been used to fully explore the 15-Puzzle with 1.4 Terabytes of harddisk in about three weeks [16]. Since harddisk operations are several times slower than the internal operations, interleaving expansion and merging through threads also accelerated the performance. It has also been used to generate very large abstract state spaces that exceed main memory capacity [27].

4 External Search in Real-Time Systems

One of the involved differences between real-time reachability and ordinary reachability analysis is the *inclusion-check*. While in (delayed) duplicate elimination we omit all identical states from further consideration, in real-time model checking we have to check inclusions of the form $Z \subseteq Z'$ to detect duplicate states. Once Z is *closed under entailment*, in the sense that no constraint of Z can be strengthened without reducing the solution set, the time-complexity for inclusion checking is linear to the number of constraints in Z.

Subsequently, while porting real-time model checking algorithms to an external setting, we have to provide an option for the elimination of zones. Since we cannot define a *total order* on zones, trivial external sorting schemes are useless in our case. In our proposal of External Breadth First Searchwe exploit the fact that two states (l, Z) and (l', Z') are comparable only when $l = l'$. This motivates the definition of *zone union* $\mathcal{U}$ where all zones correspond to the states sharing a common discrete part l, and for all $Z, Z' \in \mathcal{U}$, we have $Z \nsubseteq Z'$.

Duplicate states can now be removed by first sorting with respect to the discrete part l, which will bring all states sharing the same l close together, and then doing a one-to-one comparison among all such states. The result of this phase is a file where states are sorted according to the discrete parts l forming duplicate free zone unions.

However, the one-to-one comparison of all the zones for a particular l can only be performed I/O-efficiently when all the states sharing the same l can be read into the main memory. Throughout this presentation, we assume that this requirement holds. The same approach of internalizing zone unions is available during set refinement with respect to predecessor files. We load both the zone union from the predecessor file and the one in the unrefined file and check for the entailment condition.

State spaces that appear in model checking are usually directed and hence just removing duplicates with respect to the last previous two layers is not sufficient. The crucial complexity parameter is the locality or duplicate elimination scope as defined in [26], which defines the number of previous levels to be considered. In the text, this notion of locality for an automaton A is referred to as *locality*(A). Let Z_0 denotes the

```
Procedure External Breadth First Search
Input: A timed automata $A = (\mathcal{S}, l_0, \mathcal{R}, Inv, \mathcal{T})$; a symbolic initial state $(l_0, Z_0)$.
begin
  $Open(0) \leftarrow \{(l_0, Z_0)\}$                        ;; START WITH THE INITIAL STATE
  $j \leftarrow 1$
  while $(Open(j-1) \neq \emptyset)$
    $A(j) \leftarrow Succ(Open(j-1))$
    forall $(l, Z) \in A(j)$                                 ;; ITERATE ON ALL SUCCESSORS
      if $(l \cap \mathcal{T} \neq \emptyset)$                                    ;; GOAL FOUND
        return $ConstructSolution()$                              ;; RETURN SOLUTION
    $A'(j) \leftarrow$ remove redundant zones within $A(j)$        ;; DUPLICATES WITHIN THE LAYER
    for $loc \leftarrow 1$ to $locality(A)$                  ;; DUPLICATES SEEN IN PREVIOUS LAYERS
      $A''(j) \leftarrow A'(j) \setminus$
        $\{(l, Z') \in Open(j - loc) \mid (l, Z) \in A'(j), Z \subseteq Z'\}$
    $Open(j) \leftarrow A''(j)$
    $j \leftarrow j + 1$
end
```

Fig. 1. External Breadth First Search: (l_0, Z_0) is the initial state of the timed automaton A and $\mathcal{T}$ are the desired goal states.

empty zone. The locality of a directed search graph with (l_0, Z_0) being the start state is defined as

$$\max\{\delta((l_0, Z_0), (l, Z)) - \delta((l_0, Z_0), (l', Z'))\} + 1$$

for all states (l, Z), (l', Z'), with (l', Z') being a successor of (l, Z) and δ being the shortest-path distance between two states. For undirected graphs the above equation evaluates to 2 – validating the proof of Munagala and Ranade.

In Figure 1, we depict the pseudo-code of the algorithm that performs an External Breadth First Search on real-time systems with *symbolic states* representation. There is no hash-table involve in the algorithm but we rely on alternative duplicates removal techniques. Starting with the initial state, the algorithm performs generates all the nodes of layer $j-1$ generating the successors in layer $j-1$. Duplicates are removed in two steps: removing all the redundant zones from within a layer, and wrt. *locality*(A) many previous layers. The sets A, A', and A'' act as temporary sets. Each set is mapped to a file and a corresponding internal memory buffer. New states are first inserted into the buffer and flushed to the file once the buffer is full.

For a timed automaton A with $\mathcal{S}$ as the set of states and $\mathcal{R}$ the set of transitions in a real-time system A, we obtain the following worst-case I/O complexity of External Breadth First Search.

Theorem 1. *For the problem of symbolic reachability in timed automata, if all zone unions individually fit into the main memory* External Breadth First Search *for can be executed in* $O(\text{sort}(|\mathcal{R}|) + \text{locality}(A) \cdot \text{scan}(|\mathcal{S}|))$ *I/Os.*

Proof. The proof extends the I/O complexity of external Breadth-First search for undirected graphs. For directed graphs, the duplicate elimination scope is equal to *locality*(A),

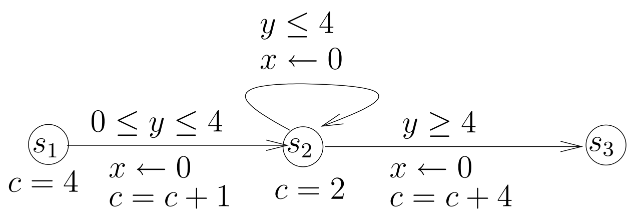

Fig. 2. Example of a priced timed automaton.

which, in turn, effects the number of layers that we have to scan in order to remove all the duplicates. □

The memory assumption is almost always fulfilled in practice, as current amounts of main memory can maintain several millions of zones. If some zone unions still fail to fit into main memory, we have to rescan the zone unions in one file again and again. If the size of the largest zone union is $\mathcal{U}_{\max}$, this will accumulate to $O(\mathit{locality}(A) \cdot \frac{|E|}{\mathcal{U}_{\max}} \cdot \mathit{scan}(\mathcal{U}_{\max})^2)$ I/Os in the worst case for checking the duplicates in the previous layer and for compacting a sorted file.

4.1 Linearly Priced Timed Automata

Linearly Priced Timed Automata (LPTA) are timed automata with (linear) cost variables. For the sake of brevity, we restrict their introduction to one cost variable c. Cost increases at states with respect to a predefined rate and in transitions with respect to an update operation. The cost-optimal reachability problem is to find a trajectory that minimizes the overall path costs. Figure 2 shows a timed automata with 3 states s_1 (*init*), s_2 (*intermediate*), s_3 (*goal*) with two clock variables x and y and the clock constraints defined on the transitions. The rate of cost variable c is 4 at s_1 and 2 at state s_2. The minimum cost of reaching location s_3 with cost 13 correspond to the trajectory $(d(0), t_1, d(4), t_2)$ of waiting 0 steps in s_1 and then taking the transition to s_2, where four time steps are spent until the transition to the goal in s_3.

Similar to the timed automata, for LPTAs we use the notion of priced zone to represent the symbolic states. Let Δ_Z be the unique clock valuation of Z such that for all $u \in Z$ and $\forall x \in C$, we have, $\Delta_Z \leq u(x)$, i.e., it represents the lowest corner of the $|C|$-dimensional hypersurface representing a zone. In the following, we Δ_Z is referred as the zone offset.

For the internal state representation, we exploit the fact that prices are linear cost hyperplanes of zones. A *priced zone* $\mathcal{Z}$ is a triple (Z, c, r), where Z is a zone, integer c describes the cost of Δ_Z and $r : C \rightarrow \mathbb{Z}$ gives the rate for a given clock. In other words, prices of zones are defined by the respective slopes that the cost function hyperplane has in the direction of the clock variable axes. Furthermore, with $f : \mathcal{Z} \rightarrow \mathbb{Z}$, we denote the cost evaluation function based on priced zones $\mathcal{Z}$. The cost value f for a given clock $x \in C$ in the priced zone $\mathcal{Z} = (Z, c, r)$ can then be computed as $c + \sum_{x \in C} r(x)(v(x) - \Delta_z(x))$. Formally, a priced timed automata can be described as follows:

Definition 2 (Linearly Priced Timed Automata [20]). *A linearly priced timed automaton $\mathcal{A}$ over clocks C is a tuple $(\mathcal{S}, l_0, \mathcal{R}, \text{Inv}, P, \mathcal{T})$, where $\mathcal{S}$ is a finite set of automata locations, $(l_0, \mathcal{Z}_0)$ is the initial state with empty priced zone $\mathcal{Z}_0$, $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{B}(C) \times \mathcal{P}(C) \times \mathcal{S}$ is the set of transitions, each consisting of a parent state, the guard on the transition, the clocks to reset and the successor state,* Inv *assigns invariants to locations, and $P : (\mathcal{S} \cup \mathcal{R}) \to I\!N$ assigns prices to the states and transitions.*

5 External Breadth First Branch-and-Boundin Priced Real-Time Systems

Until now, we have been mainly discussing external search in directed and unweighted state spaces. But, as we move towards priced real-time systems where timed automata are extended with a cost variable, we find ourselves dealing with a weighted state space. Moreover, we are no longer interested in just some path to a particular goal state, but in an optimal path with respect to our new cost variable.

In priced real-time systems, cost f is a monotonically increasing function implying that for all $(u, v) \in \mathcal{R}$, we have $f(u) \leq f(v)$. If f^* is the optimal solution cost, the following definition captures the notion of cost-optimality for a set of goals $\mathcal{T}$ and a start state $(l_0, \mathcal{Z}_0)$.

Definition 3. ***(Cost-Optimality)*** *An algorithm is* Cost-Optimal*, if and only if, it terminates with a state $t \in T$ and $f(t) = f^*$.*

In such directed and weighted graphs, BFS does not guarantee an optimal solution. A natural extension of BFS is to continue the search when a goal is found and keep on searching until a *better* goal is found or the state space is exhausted. A Branch-and-Bound (BnB) search algorithm is an extension to an uninformed search algorithm that does not stop when it finds the first goal, but instead *prunes* all the states that do not improve on the last solution cost. Given that the cost function is monotone, which is the case with f, BnB always terminates with an optimal solution.

The main traversal policy of a Branch-and-Bound algorithm can be borrowed from either breadth-first search, depth-first search, or best-first search. A Best-First BnB algorithm, though very well suited for small-sized problems can create a bottleneck for larger problems. Best-first search picks a state u such that for all $v \in$ *Open*, we have $f(u) \leq f(v)$, for the next expansion. This selection criteria calls for a much larger horizon to be saved in the memory as compared to the Breadth First Search or a Depth First Search. Moreover, both depth-first and best-first traversal policies show no locality in the way they expand states - unlike Breadth First Search , where every state in a layer j is expanded before any state of the layer $j + 1$. This property makes Breadth First Search a good candidate for branch-and-bound.

Because of being in a weighted state space, we have to pay an overhead by re-opening already seen states. Consider the following example as illustrated in Fig. 3. A Breadth-First search visits state v for the first time (top right copy) and stores it. Goal state g is also visited and its cost is saved. When the search reaches state v for the second time along a longer path (bottom left copy), but this time with a better cost, v will be pruned away while subtracting previous layers and g will never be reached. If the new

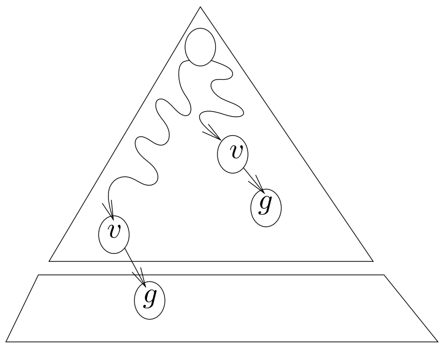

Fig. 3. Anomaly in the Breadth-First Branch-and-Bound. g is a goal state

path to g has a better cost, we lose our claim for optimality. Due to this anomaly, the duplicate detection policy has to be adapted to make it compatible with weighted state spaces. Now we are not allowed to remove a duplicate state if its cost is better than what we have seen earlier.

Definition 4. ***Duplicate state in priced domains*** *$(l, \mathcal{Z})$ is a duplicate state of $(l', \mathcal{Z}')$ if and only if $l = l'$, $\mathcal{Z} \subseteq \mathcal{Z}'$ and $f(\mathcal{Z}) \geq f(\mathcal{Z}')$.*

In Figure 4, we formulate our discussion on External Breadth First Branch-and-Bound in pseudo-code. The set *Open* represents the BFS layer and the sets A, A' and A'' are temporary variables to construct the search frontier for the next iteration. Initially the goal cost *Cost* is initialized with ∞ and a goal state with a better value is searched in the successor set $A(j)$. States with a higher value than the best goal cost are pruned and saved in $A'(j)$. In the next step, we remove redundant states based on our definition of duplicate states.

The working of the algorithm is depicted in Figure 5. On x-axis we denote the layers of Breadth First exploration. Each layer is sorted with increasing cost value. Upon arriving at the first goal t_1, the next layer is pruned to only consider the nodes that have a better cost value. The exploration terminates when the last goal t_4 with the minimal cost value is expanded and no successor of t_4 improves the cost.

The I/O complexity of External Breadth First Branch-and-Bound algorithm depends on the number of times a state is re-expanded. The worst-case scenario is when the whole state space fits into one layer and the next layer has the same states but with better cost values. The following theorem states the cost-optimality and I/O complexity of the algorithm.

Theorem 2. *For the problem of cost-optimal symbolic reachability in priced timed automata with monotonic costs, if all zone unions individually fit into the main memory, External Breadth First Branch-and-Bound is* Cost-Optimal *and can be executed in $O(D \cdot (sort(|\mathcal{R}|) + \text{locality}(\mathcal{A}) \cdot scan(|\mathcal{S}|)))$ I/Os, where D is the maximal depth explored.*

```
Procedure External Breadth First Branch-and-Bound
Input: A linearly priced timed automaton $\mathcal{A} = (\mathcal{S}, l_0, \mathcal{R}, Inv, P, \mathcal{T})$;
       A symbolic initial state $(l_0, \mathcal{Z}_0)$.
begin
  $Cost \leftarrow \infty; j \leftarrow 1$                                   ;; BEST GOAL COST IS $\infty$
  $Open(0) \leftarrow \{(l_0, \mathcal{Z}_0)\}$                            ;; START WITH THE INITIAL STATE
  while $(Open(j-1) \neq \emptyset)$
    $A(j) \leftarrow Succ(Open(j-1))$
    forall $(l, \mathcal{Z}) \in A(j)$                                  ;; ITERATE ON ALL SUCCESSORS
      if $(l \cap \mathcal{T} \neq \emptyset \wedge f(\mathcal{Z}) < Cost)$                      ;; ANOTHER GOAL FOUND
        $Cost \leftarrow f(\mathcal{Z})$                                       ;; COST OF THE NEW GOAL
    $A'(j) \leftarrow A(j) \setminus \{(l, \mathcal{Z}) \in A(j) \mid f(\mathcal{Z}) \geq Cost\}$          ;; PRUNE THE EXPENSIVE STATES
    $A''(j) \leftarrow$ remove redundant zones within $A'(j)$          ;; DUPLICATES WITHIN THE LAYER
    for $loc \leftarrow 1$ to $locality(\mathcal{A})$                    ;; DUPLICATES SEEN IN PREVIOUS LAYERS
      $A''(j) \leftarrow A''(j) \setminus$
        $\{(l, \mathcal{Z}') \in Open(j - loc) \mid (l, \mathcal{Z}) \in A''(j), \mathcal{Z} \subseteq \mathcal{Z}' \wedge f(\mathcal{Z}) \geq f(\mathcal{Z}')\}$
    $Open(j) \leftarrow A''(j)$
    $j \leftarrow j + 1$
  if $(Cost \neq \infty)$
    return $ConstructSolution()$                          ;; CONSTRUCT SOLUTION IF FOUND
end
```

Fig. 4. External Breadth First Branch-and-Bound: $(l_0, \mathcal{Z}_0)$ is the symbolic initial state of the graph $\mathcal{A}$ and $\mathcal{T}$ are the desired goal states.

Proof. Since External Breadth First Branch-and-Bound expands at least all states $(l, \mathcal{Z})$ with $f(l, \mathcal{Z}) < f^*$, the algorithm terminates with the optimal solution. The *I/O* complexity of the algorithm is inherited from the External Breadth First Search search (cf. Theorem 1). The factor D is introduced due to re-openings. $\square$

Furthermore, we can say that if there are several goal states in the state space with different solution costs, then an External Breadth First Branch-and-Bound run will explore at most as many states as a complete External Breadth First Search run.

Lemma 1. *If m is the number of states expanded by External Breadth First Branch-and-Bound and n is the number of states expanded by a complete exploration of External Breadth First Search, then $m \leq n$.*

Proof. External Breadth First Branch-and-Bound does not change the order in which states are looked at during a complete External Breadth First Search exploration. There can be two cases:

1. $|\mathcal{T}| = 1$: There exist just one goal state t which is also the last state in a breadth-first search tree. For this case clearly $n = m$.
2. $|\mathcal{T}| > 1$: There exists more than one goal state in the search tree. Let $t_1, t_2 \in \mathcal{T}$ be the two goal states with $f(t_1) > f(t_2) = f^*$ and *depth*$(t_1) <$ *depth*(t_2). Since t_1 will be expanded first, $f(t_1)$ will be used as the pruning value during the next

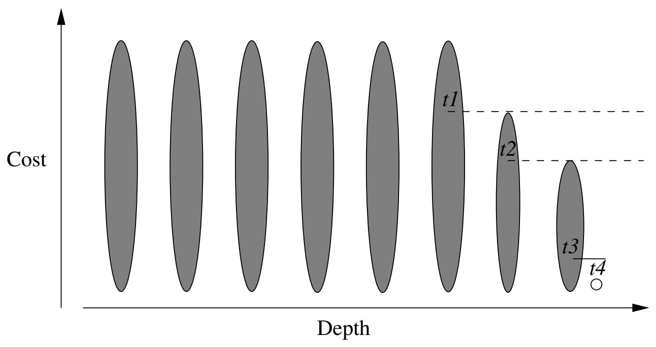

Fig. 5. A sample run of External Breadth First Branch-and-Bound; the t_i's represent different goals.

iterations. In case, there does not exists any state u in the search tree between t_1 and t_2 with $f(u) > f(t_2)$, $n = m$, else $m < n$.

□

The behaviour of External Breadth First Branch-and-Bound largely depends on how fast it reaches to some solution so that it can use that solution cost to further prune away the search space. Their exists a very trivial solution to this problem where the user provides some upper bound U on the solution cost that can be used for pruning. In case the upper bound U is actually equal to the optimal solution cost f^*, the algorithm is trivially *Cost-Optimal*.

Lemma 2. *External Breadth First Branch-and-Bound with $U = f^*$ is* Cost-Optimal.

Since the cost function f in our real-time domain is monotonically increasing, i.e., for all $(u, v) \in \mathcal{R}$, we have $f(u) \leq f(v)$, we will never prune any node that can ultimately take us to the goal node.

6 Iterative Broadening External Breadth First Branch-and-Bound

We observe that the efficiency of External Breadth First Branch-and-Bound is inversely proportional to the factor $U - f^*$. The more realistic the upper bound is, the bigger the pruning and, hence, the lesser the number of expansions. This observation guides us to an iterative strategy to find a good upper bound. We suggest to use only the first $k\%$ of the states when sorted with respect to the increasing cost value and discard the rest of the states in the layer. Hopefully, the algorithm will terminate with a solution, giving us a good upper bound on the optimal solution cost. Using the found solution cost as the upper bound for an increased value of k, we hope to converge to optimal solution cost when k approaches to 100. We will refer the parameter k as the beam width.

Unfortunately, there is an apparent problem with this approach. It is possible that for a particular iteration we arrive at a goal state, but at the next iteration we do not.

This problem is more frequent in real-time domains, where there can be many different states with the same f-value, residing in a set that has no total order. The algorithm is not guaranteed to converge with increasing k (exception is when $k = 100\%$ and the whole state space is considered). Let k_i be the value of k in the ith iteration. For the algorithm to converge, the coverage area of the $(i+1)$th iteration must be at least as large as the coverage area of the ith iteration. Formally, for any layer j,

$$\mathit{Open}_i(j) \subseteq \mathit{Open}_{i+1}(j) \tag{1}$$

Such a guarantee can only be given if the maximum cost value that was chosen in the $(i+1)$th iteration for layer j is greater than or equal to the maximum cost value chosen in the i-th iteration. For Condition 1 to hold throughout the exploration, we propose the following selection criterion.

Selection Criterion the best $k\%$ states of a layer *plus* all the states that have the same f-value as that of the last state of the selected list *plus* all the states that have the smaller f-value as that of the maximum selected f-value of the last iteration.

With this selection criterion, for a particular cost f', we either choose all the states with a f value equal to f' or choose none.

Figure 6 shows the pseudo-code for the actual exploration involving upper bound pruning and the above mentioned selection criteria. The parameters of the algorithms are the beam width k (in percent), the upper bound U and the vector $F_{\max}$ of maximal f-values from the last iteration. With successive iterations, the value of k is increased and the solution cost value of the previous iteration is used as an upper bound. The set *Open* denotes the search frontier, sliced into layers as before. The sets A, A' and A'' are temporary sets, to construct the search frontier for the next iteration. Both the new *Cost* and the new vector of maximal f-values are returned. We use π_n to denote the n-th element in the sorted permutation of a set.

6.1 Correctness

Let U_i' be the cost of the solution found by Iterative Broadening External Breadth First Branch-and-Bound in the ith iteration with $k = k_i$ and $U = U_i$ as the arguments. In the following, we show that the algorithm converges for increasing value of k.

Lemma 3. *The selection criterion for Iterative Broadening External Breadth First Branch-and-Bound guarantees the coverage condition for every iteration i.*

Proof. We prove it by induction on the layer j. For $j = 0$, $\mathit{Open}_i(0) \subseteq \mathit{Open}_{i+1}(0)$. Assume that it holds for layer $j-1$ i.e, $\mathit{Open}_i(j-1) \subseteq \mathit{Open}_{i+1}(j-1)$. Generating the successor sets for both sides of the relation yields $\mathit{Succ}(\mathit{Open}_i(j-1)) \subseteq \mathit{Succ}(\mathit{Open}_{i+1}(j-1))$. Removing duplicates from the successor sets on both sides does not change the subset condition. Now we turn to pruning. The selection criteria guarantees that the values $F^j_{\max}$ increase monotonically for increasing value of i, i.e., $F^j_{i,\max} \leq F^j_{i+1,\max}$. Moreover cost plateaux are completely searched. Therefore, pruning does not change the subset condition, so that $\mathit{Open}_i(j) \subseteq \mathit{Open}_{i+1}(j)$. $\square$

```
Procedure Iterative Broadening External Breadth First Branch-and-Bound($k, U, F_{\max}$)
Input: A linearly priced timed automaton $\mathcal{A} = (\mathcal{S}, l_0, \mathcal{R}, Inv, P, \mathcal{T})$;
       A symbolic initial state $(l_0, \mathcal{Z}_0)$.
begin
  $Cost \leftarrow U$; $j \leftarrow 1$                                   ;; BEST GOAL COST IS $U$
  $Open(0) \leftarrow \{(l_0, \mathcal{Z}_0)\}$                         ;; ALWAYS SART WITH THE INITIAL STATE
  while $(Open(j-1) \neq \emptyset)$
    $A(j) \leftarrow Succ(Open(j-1))$
    forall $(l, \mathcal{Z}) \in A(j)$                                 ;; ITERATE ON ALL SUCCESSORS
      if $(l \cap \mathcal{T} \neq \emptyset \wedge f(\mathcal{Z}) < Cost)$                 ;; ANOTHER GOAL FOUND
        $Cost \leftarrow f(\mathcal{Z})$                                         ;; COST OF THE NEW GOAL
    $A'(j) \leftarrow A(j) \setminus \{(l, \mathcal{Z}) \in A(j) \mid f(\mathcal{Z}) \geq Cost\}$         ;; PRUNE THE EXPENSIVE STATES
    $A''(j) \leftarrow$ remove redundant zones within $A'(j)$          ;; DUPLICATES WITHIN THE LAYER
    for $loc \leftarrow 1$ to $locality(\mathcal{A})$                      ;; DUPLICATES SEEN IN PREVIOUS LAYERS
      $A''(j) \leftarrow A''(j) \setminus$
        $\{(l, \mathcal{Z}') \in Open(j - loc) \mid (l, \mathcal{Z}) \in A''(j), \mathcal{Z} \subseteq \mathcal{Z}' \wedge f(\mathcal{Z}) \geq f(\mathcal{Z}')\}$
    $A''(j) \leftarrow$ External-sort $A''(j)$ w.r.t the cost function $f$
    $n \leftarrow \lfloor (k \cdot |A''(j)|)/100 \rfloor$               ;; THERE ARE $n$ MANY STATES IN THE BEST $k\%$
    $(l_n, \mathcal{Z}_n) \leftarrow \pi_n(A''(j))$                                   ;; PICK THE $n$-TH STATE
    $F^j_{\max} \leftarrow \max\{F^j_{\max}, f(\mathcal{Z}_n)\}$      ;; COMPUTE THE NEW MAX $F$ VALUE FOR THE LAYER
    $Open(j) \leftarrow \{(l, \mathcal{Z}) \in A''(j) \mid f(\mathcal{Z}) \leq F^j_{\max}\}$           ;; KEEP ONLY THE best STATES
    $j \leftarrow j + 1$
  if $(Cost < U)$                  ;; IF THE BOUND HAS IMPROVED CONSTRUCT THE SOLUTION
    ConstructSolution()
  return $Cost, F_{\max}$                                          ;; RETURN NEW UPPER BOUND
end
```

Fig. 6. Iterative Broadening External Breadth First Branch-and-Bound. k represents beam width, U the upper bound, and $F_{\max}$ represents the maximum cost used in each layer during the last iteration. $(l_0, \mathcal{Z}_0)$ is the symbolic initial state of the priced timed automata $\mathcal{A}$ and $\mathcal{T}$ are the desired goal states.

Lemma 4. *For all iterations i in Iterative Broadening External Breadth First Branch-and-Bound, we have $U'_{i+1} \leq U'_i$.*

Proof. Since the coverage area of iteration $i + 1$ is larger than the coverage area of iteration i, in the worst case it does not improve on the solution quality i.e., $U'_{i+1} = U'_i \leq U_i$, else we have $U'_{i+1} \leq U'_i \leq U_i$. In both cases, $U'_{i+1} \leq U'_i$. □

Theorem 3. *Iterative Broadening External Breadth First Branch-and-Bound converges to the optimal solution.*

Proof. Lemma 3 provides the necessary ground for the coverage of whole state space, which implies the completeness of the algorithm and Lemma 4 provides the convergence to the optimal solution cost that proves its optimality. □

7 Experiments

We have implemented the algorithms External Breadth First Branch-and-Bound, and Iterative Broadening External Breadth First Branch-and-Bound on top of UPPAAL CORA. Our implementation also extends UPPAAL making it capable to perform External Breadth First Search in timed automata. The main memory requirements are kept constant[4]. Hash tables are replaced by files on harddisk with a small internal buffer for I/O efficiency. As the maximum file size on most file systems is 2GB, we also provide large file support, that splits files if they become too large. Trails for found solutions are reconstructed by saving the predecessor together with every state, by using backtracking along the stored files, and by looking for matching predecessors. This results in a I/O complexity that is at most linear to the number of stored states.

A limited functionality (which nonetheless does not compromise the correctness of the approach) of the current implementation is on the duplicate detection scope and on external sorting. We remove duplicates from the internal buffer before flushing it but the duplicates within different flushed buffers are not merged. All experiments are run on a Pentium-4 with 150 GB of harddisk space and 2GB RAM running Linux. We chose different instances of aircraft landing scheduling (ALS), for which [5] presented a UPPAAL CORA model. It involves considering a timed automaton for each of the airplane and runways.

We start with a smaller instance involving just 1 runway and 10 planes. Table 1 (left) provides the results of running Iterative Broadening External Breadth First Branch-and-Bound. Here k denotes the coverage, U the initial bound and U' the optimal solution obtained. The behaviour of pruning on the number of expanded states is quite evident. We also see a converging behaviour of the algorithm. In the last row we report the results for External Breadth First Branch-and-Bound to show the effect of pruning on the search space. Our result matches with the one found by UPPAAL CORA. Table 1 (right) illustrates the results for the instance, where we created two independent automata for runways and planes. We then instantiated 1 runway and 10 planes from the first type and 1 runway and 10 planes from the other. UPPAAL CORA with internal BnB cannot solve the instance because of memory requirements. Being an exact dual, the solution has to be 1400, which validates our implementation. With Iterative Broadening, we were able to find an optimal solution. On the other hand, External Breadth First Branch-and-Boundcould not finalize its execution in two hours consuming about 3 GB with 280 bytes per state, while expanding depth 19 - optimal solution lies at depth 40. The process was manually killed.

For the third instance, we chose another instance of aircraft scheduling problem that was obtained by a translation from PDDL planning models [7]. The internal version of UPPAAL CORA failed to reach any solution for 3 planes and after quickly consuming about 1.6 GB of main memory started to swap on harddisk. For this instance just for 3 planes a total of 13 clocks were used. Our iterative broadening strategy, for $k < 100$ didn't produce any solution. For $k = 100$, the algorithm ran for about 12 hours consuming a total of 311 GB and ran out of harddisk space using a mere 2KB per state. On a harddisk with just 150 GB available, this was achieved by removing the

[4] Up to a leak of at most 100 MB per hour.

k	U	U'	$Expanded$
1	∞	970	91
20	970	970	91
40	970	810	125
60	810	710	281
80	710	700	439
100	700	700	577
100	∞	700	31,458

k	U	U'	$Expanded$
0.1	∞	1940	1,060
20	1940	1940	1,285
40	1940	1420	18142
60	1420	1410	69,341
80	1410	1410	147,128
100	1410	1400	195,145
100	∞	—	—

Table 1. ALS with 1 runway and 10 planes (left), and with 2 runways and 20 planes (right).

previous layers manually. Up till the 40th layer there was no solution. In Fig. 7, we depict the graph where space consumption for each layer is shown. The internal size of the program remained under 1.8 GB.

8 Conclusion

We have seen an approach for large scale scheduling based on external exploration on priced timed automata. We contributed two algorithms: *External Breadth First Branch-and-Bound* and *Iterative Broadening External Breadth First Branch-and-Bound.* Both algorithms perform an external Breadth First Search on the search space and preserve optimality of the computed cost values. Having performed an exploration of more than a quarter of a Terabyte, we believe to have pushed the limits of practical scheduling and model-checking in real-time domains.

The exploration can be performed on multiple disks, as sorting and searching can be distributed with optimal I/O efficiency. As external exploration realizes a controlled streamed access to states, there is also potential for a parallel implementation. A parallel and distributed reachability checking algorithm of UPPAAL based on the *Message Passing Interface* (MPI) partitions the list of explored states using a simple hash function [4]. It restricts itself to blind exploration.

We have not talked about heuristic search, although the UPPAAL CORA models incorporate hand-coded search heuristics to accelerate the exploration. A recent proposal to generate heuristics for UPPAAL automatically has recently been provided by [19].

Iterative Broadening has been introduced by [10]. The Breadth First BnB approach is related to Breadth-First Heuristic Search (BFHS) [26], a frontier search method that was designed to save internal memory. It is based on the observation that the Breadth First Search frontier is often much smaller than the best-first search frontier. A recent extension of BFHS is its integration with beam search known as Beam-Stack Search [28]. As it iterates on different beams, this algorithm is a natural competitor for Iterative Broadening External Breadth First Branch-and-Bound. This algorithm is also guaranteed to continously converge. There are several differences to our approach. The beam width in Beam-Stack Search is driven by the limits of main memory (previous layers can be flushed to the harddisk). Such a limit is not needed in our case, as we exploit the secondary storage. Therefore, we introduce parameter k to control the beam width. Moreover, a backtracking strategy is employed to pick more elements from the previous layer in case the upper bound is not improved.

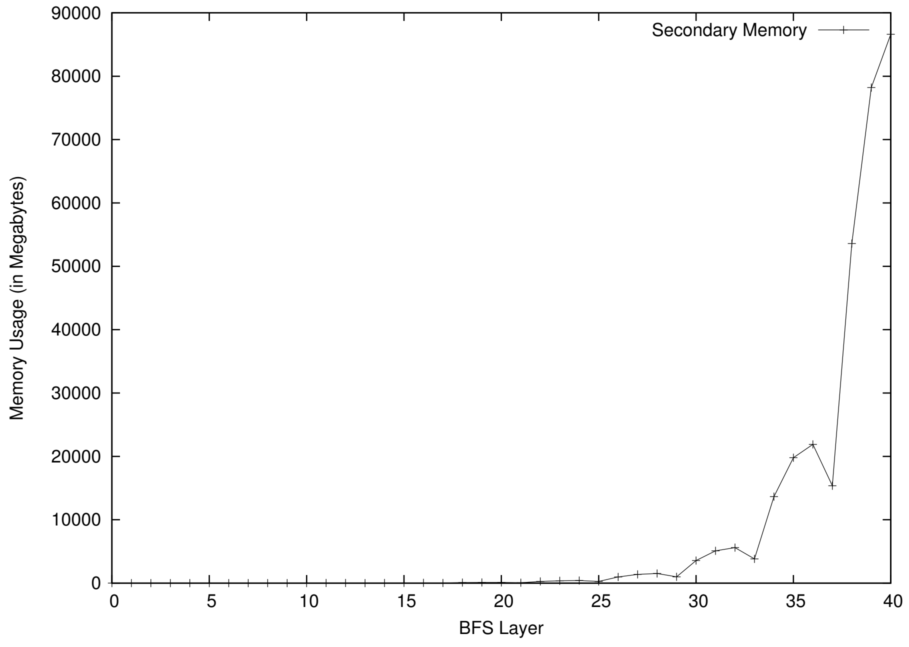

Fig. 7. Space consumption for each BFS Layer.

The approach we are currently working on, splits the layer that is being expanded, into several ones, and distributes the work among different processors. As states can be expanded independently of each other, a speedup is expected.

References

1. A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Journal of the ACM*, 31(9):1116–1127, 1988.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
3. L. Arge, M. Knudsen, and K. Larsen. Sorting multisets and vectors in-place. In *Workshop on Algorithms and Data Structures (WADS)*, LNCS, pages 83–94, 1993.
4. G. Behrman, A. Fehnker, and F. Vaandrager. Distributed timed model checking - how the search order matters. In *CAV*, 2000.
5. G. Behrmann, K. G. Larsen, and J. I. Rasmussen. Optimal scheduling using priced timed automata. In *ICAPS Workshop on Verification and Validation of Model-Based Planning and Scheduling Systems*, 2005.
6. F. Cassez, A. David, E. Fleury, K. G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, 2005.
7. H. Dierks. Finding optimal plans for domains with restricted continuous effects with cora. In *ICAPS Workshop on Verification and Validation of Model-Based Planning and Scheduling Systems*, 2005.

8. S. Edelkamp and S. Jabbar. Large-scale directed model checking LTL. In *Model Checking Software, SPIN*, pages 1–18, 2006.
9. S. Edelkamp, S. Jabbar, and S. Schroedl. External A*. In *German Conference on Artificial Intelligence (KI)*, pages 226–240, 2004.
10. M. Ginsberg and W. Harvey. Iterative broadening. *Artificial Intelligence*, pages 367–383, 1992.
11. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *ACM STOC*, pages 373–381, 1995.
12. D. S. Hirschberg. A linear space algorithm for computing common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.
13. S. Jabbar and S. Edelkamp. I/O efficient directed model checking. In *VMCAI*, 2005. 313–329.
14. S. Jabbar and S. Edelkamp. Parallel external directed model checking with linear I/O. In *VMCAI*, pages 237–251, 2006.
15. R. E. Korf. Divide-and-conquer bidirectional search: First results. In *IJCAI*, pages 1184–1191, 1999.
16. R. E. Korf and P. Schultze. Large-scale parallel breadth-first search. In *AAAI*, pages 1380–1385, 2005.
17. R. E. Korf and W. Zhang. Divide-and-conquer frontier search applied to optimal sequence allignment. In *AAAI*, pages 910–916, 2000.
18. L. Kristensen and T. Mailund. Path finding with the sweep-line method using external storage. In *ICFEM*, pages 319–337, 2003.
19. S. Kupferschmid, J. Hoffmann, H. Dierks, and G. Behrmann. Adapting an AI planning heuristic for directed model checking. In *SPIN*, pages 35–52, 2006.
20. K. G. Larsen, Gerd Behrmann, E. Brinksma, A. Fehnker, T. S. Hune, P. Petterson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *CAV*, pages 493 – 505, 2001.
21. K. G. Larsen, F. Larsson, P. Petterson, and W. Yi. Efficient verification of real-time systems: Compact data structures and state-space reduction. In *IEEE Real Time Systems Symposium*, pages 14–24, 1997.
22. K. Munagala and A. Ranade. I/O-complexity of graph algorithms. In *SODA*, pages 687–694, 1999.
23. J. I. Rasmussen, K. G. Larsen, and K. Subramani. Resource-optimal scheduling using priced timed automata. In *TACAS*, pages 220–235, 2004.
24. P. Sanders, U. Meyer, and J. F. Sibeyn. *Algorithms for Memory Hierarchies*. Springer, 2002.
25. U. Stern and D. Dill. Using magnetic disk instead of main memory in the murphi verifier. In *International Conference on Computer Aided Verification (CAV)*, pages 172–183, 1998.
26. R. Zhou and E. Hansen. Breadth-first heuristic search. In *ICAPS*, pages 92–100, 2004.
27. R. Zhou and E. Hansen. External-memory pattern databases using structured duplicate detection. In *AAAI*, 2005.
28. R. Zhou and E. A. Hansen. Beam-stack search: Integrating backtracking with beam search. In *ICAPS*, pages 90–98, 2005.