

Reasoning about Online Algorithms with Weighted Automata

Benjamin Aminof*

Orna Kupferman*

Robby Lampert*

Abstract

We describe an automata-theoretic approach for the competitive analysis of *online algorithms*. Our approach is based on *weighted automata*, which assign to each input word a cost in $\mathbb{R}^{\geq 0}$. By relating the “unbounded look ahead” of optimal offline algorithms with nondeterminism, and relating the “no look ahead” of online algorithms with determinism, we are able to solve problems about the competitive ratio of online algorithms, and the memory they require, by reducing them to questions about *determinization* and *approximated determinization* of weighted automata.

1 Introduction

In *formal verification*, we verify that a system has a desired property by checking whether a mathematical model of the system satisfies a formal specification of the property. Early work on formal verification handled finite-state hardware designs. Current work already copes with infinite-state software systems, complex distributed systems, and so on [2, 15], and is widely and successfully used in the industry [9]. An important feature of formal verification is that rather than reasoning only about input/output relations of terminating systems (for example, the output is the gcd of the two numbers in the input), it enables reasoning about *reactive systems*, which maintain an on-going interaction with their environment. For example, one can check that an operating system never reaches a deadlock, or that every request in some communication protocol is eventually acknowledged.

In this work we extend the scope of formal verification to reasoning about *online algorithms*. An online algorithm can be viewed as a reactive system: at each round, the environment issues a request, and the algorithm should process it. The sequence of requests is not known in advance, and the goal of the algorithm is to minimize the overall cost of processing all the requests in the sequence. Online algorithms for many problems have already been extensively studied for several decades, and have aroused much interest, both from a practical and a theoretical point of view [4].

While the interaction described above between an online algorithm and its environment is at the heart of formal verification, the questions that are traditionally answered by formal verification techniques are very different from those that are asked in the context of online algorithms. In formal verification, a system is checked with respect to a given specification. On the other hand, the most interesting question about an online algorithm refers to its *competitive ratio*: the worst-case (with respect to all input sequences) ratio between the cost of the algorithm and the cost of an optimal solution (one that may be given by an *offline algorithm*, which knows the input sequence in advance). While current formal verification techniques can check qualitative properties of an online algorithm (e.g., “whenever a request to a page is made, and this page is not in the cache, the page is brought into the cache”) and even answer quantitative questions about it (e.g., “what is the maximal number of page faults within a window of k rounds?”) [8], current techniques cannot refer to the optimal solution, and hence, they cannot reason about the competitive ratio. Likewise, while synthesis algorithms and tools successfully generate systems that satisfy a given specification [16], current synthesis algorithms cannot, for example, synthesize (or decide that there does not exist) online algorithms that are as good, or competitive with some given ratio, as a given offline algorithm.

Our approach to formally reasoning about online algorithms is based on *weighted finite automata* (WFAs, for short) [18, 20]. Essentially, we relate the “unbounded look ahead” of the optimal offline algorithm with nondeterminism, and relate the “no look ahead” of online algorithms with determinism. This enables us to reduce questions about the competitive ratio of online algorithms to questions about *determinization* and *approximated determinization* of WFAs. Below we further elaborate on our approach and our results.

A WFA \mathcal{A} induces a partial *cost* function from Σ^* to $\mathbb{R}^{\geq 0}$. Technically, each transition of \mathcal{A} has a cost, the cost of a run is the sum of the costs of the transitions taken along the run, and the cost of a word w , denoted $\text{cost}(\mathcal{A}, w)$, is the minimum cost over all accepting runs on it (the cost is undefined if no run on the word is accepting). Consider an optimization problem P with

*School of Computer Science and Engineering, Hebrew University, Israel. E-mail: {benj,orna,robil}@cs.huji.ac.il.

requests in Σ . An algorithm for P can be viewed as a mapping of words in Σ^+ to a set of actions available to the algorithm [3]. For a finite set S of configurations, we say that an algorithm uses memory S if there is a regular mapping of Σ^* into S such that the algorithm behaves in the same manner on identical continuations of words that are mapped to the same configuration.

The set of online algorithms for P that use memory S induces a WFA \mathcal{A}_P , with alphabet Σ and state space S , such that the transitions of \mathcal{A}_P correspond to actions of the algorithms and the cost of each transition is the cost of the corresponding action. We argue that many optimization problems have algorithms that use finite memory. We demonstrate this on the paging, k -server, ski-rental, load-balancing, and Δ -paid exchange static list accessing problems.

Given a finite sequence of requests $w \in \Sigma^*$, each run of \mathcal{A}_P on w corresponds to a way of serving the requests in w by an algorithm with memory S . The set of all runs include all such ways, thus $\text{cost}(\mathcal{A}, w)$ is the cost of an optimal offline algorithm on w that uses memory S . On the other hand, an online algorithm has to process each request as soon as it arrives. Hence, an online algorithm corresponds to a deterministic automaton *embodied* in \mathcal{A}_P . Indeed, for every configuration $s \in S$ and request $\sigma \in \Sigma$, the algorithm suggests a particular way to process σ from s , inducing a single transition labeled σ from s . Accordingly, there exists an online algorithm for P that performs as well as the optimal offline algorithm iff \mathcal{A}_P embodies an equivalent deterministic automaton, in which case we say that \mathcal{A}_P is *determinizable by pruning*. Similarly, there exists an α -competitive online algorithm for P , for $\alpha > 1$, iff \mathcal{A}_P embodies a deterministic automaton \mathcal{A}'_P that α -approximates \mathcal{A}_P (the automaton \mathcal{A}'_P accepts the same set of words as \mathcal{A}_P , and $\text{cost}(\mathcal{A}'_P, w) \leq \alpha \cdot \text{cost}(\mathcal{A}_P, w)$ for all words w in this set). Then, we say that \mathcal{A}_P is *α -determinizable by pruning*.

Restricting the determinization procedure to automata embodied in \mathcal{A}_P guarantees that transitions in the automaton still correspond to actions of an algorithm for P . An online algorithm, however, may require more memory than an offline algorithm for the same problem. For example, in the paging problem, an offline algorithm only has to remember in each round the set of pages that are in the cache, whereas known online algorithms that achieve the best competitive ratio are marking algorithms, which remember, in addition, some order on the pages in the cache, or some other information. To address this point, we also consider a variant of determinization by pruning that allows a refinement of the state space of \mathcal{A}_P before pruning it to a deterministic automaton. We show that such a refinement

indeed corresponds to an extension of the memory used by the algorithm.

We study the problems of deciding whether a WFA is determinizable (or α -determinizable) by pruning, with and without refinement. The problems are, in fact, challenging already for the unweighted case, and we first solve them in this setting¹. We show that the problem of deciding whether a WFA is determinizable by pruning can be solved in polynomial time. Our algorithm makes use of the local nature of pruning – each state should have, for each input letter $\sigma \in \Sigma$, a σ -transition that “covers all other σ -transitions”. The local nature of pruning, however, cannot be used when considering approximation, and we show that the problem of deciding whether a WFA is α -determinizable by pruning, for $\alpha > 1$, is NP-complete. It follows that given an optimization problem P and a finite set S of configurations, the problem of deciding whether there is an online algorithm for P with configurations in S , that is as good as an offline algorithm with configurations in S , can be solved in polynomial time. On the other hand, the problem of deciding whether there is an online algorithm for P with configurations in S that is α -competitive, for a fixed $\alpha > 1$, with respect to an offline algorithm with configurations in S , is NP-complete.

The complications that approximation brings with it are carried over to the setting in which an extension of the memory is allowed. We prove that while extending the memory cannot help an online algorithm to perform as well as the offline algorithm (that is, if an offline algorithm uses memory S , and no 1-competitive online algorithm with configurations in S exists, then there is no 1-competitive online algorithm at all), memory may help in order to decrease a competitive ratio $\alpha > 1$ (that is, there are problems for which an offline algorithm uses configurations in S , no online algorithm with configurations in S is α -competitive, but there is an online algorithm with richer configurations that is α -competitive)².

In Section 6, we discuss the practical aspects of implementing our framework. In particular, we discuss symbolic approaches that cope with the large state space that our algorithms handle, and parametric methods, which allow to reason about a system with many identical processes by studying properties of one of the processes.

¹The problem of determinization by pruning is of interest also in the unweighted case. As described in [14], automata that are determinizable by pruning can be used in the process of synthesis and game solving.

²We note that while it is widely believed that a k -competitive online algorithm for the paging problem needs more memory than the optimal offline algorithm, this is not the case [11].

1.1 Related work Our automata-theoretic approach for reasoning about online algorithms adopts and extends ideas from work done in the formal-verification community. An automata-theoretic approach for reasoning about systems and their specifications has been suggested in [22], and has been extensively studied and implemented since then. As discussed above, the known approach is not suitable for reasoning about online algorithms. Determinization of WFA is studied in [20], but the technique and the applications are different from those of determinization by pruning, which we study here.

The online-algorithms community has studied several abstract models for competitive analysis. The ongoing interaction that takes place in online algorithms can be modeled, for example, by means of *games in strategic form* [17] and *request-answer games* [3]. Other work considers models for specific problems (e.g., [1] for paging). The model that is closest to our automata-theoretic approach is the one of *metrical task systems* [5, 19]. The expressive power and the applications of the various models are different, however, from our weighted automata.

A full version of the paper can be found in the authors' web sites.

2 Preliminaries

2.1 Weighted automata Standard automata map words in Σ^* to either “accept” or “reject”. A weighted automaton can be viewed as a partial function (defined only for accepted words) from Σ^* to $\mathbb{R}^{\geq 0}$. Formally, a *weighted finite automaton* (WFA, for short) is $\mathcal{A} = \langle \Sigma, Q, \Delta, c, Q_0, F \rangle$, where Σ is a finite input alphabet, Q is a finite set of states, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $c : \Delta \rightarrow \mathbb{R}^{\geq 0}$ is a cost function, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of final states. A transition $d = \langle q, a, p \rangle \in \Delta$ (also written as $\Delta(q, a, p)$) can be taken when reading the input letter a , and it causes \mathcal{A} to move from state q to state p with cost $c(d)$. The transition relation Δ induces a transition function $\delta : Q \times \Sigma \rightarrow 2^Q$ in the expected way. Thus, for a state $q \in Q$ and a letter $a \in \Sigma$, we have $\delta(q, a) := \{p : \Delta(q, a, p)\}$. A WFA \mathcal{A} may be nondeterministic in the sense that it may have many initial states, and that for some $q \in Q$ and $a \in \Sigma$, it may have $\Delta(q, a, p_1)$ and $\Delta(q, a, p_2)$, with $p_1 \neq p_2$. If $|Q_0| = 1$ and for every state $q \in Q$ and letter $a \in \Sigma$ we have $|\delta(q, a)| \leq 1$, then \mathcal{A} is a *deterministic* weighted finite automaton (DWFA, for short).

For a word $w = w_1 \dots w_n \in \Sigma^*$, a run of \mathcal{A} on w is a sequence $r = r_0 r_1 \dots r_n \in Q^+$, where $r_0 \in Q_0$ and for every $1 \leq i \leq n$, we have $\langle r_{i-1}, w_i, r_i \rangle \in \Delta$. The run r is accepting if $r_n \in F$. The word w is accepted by \mathcal{A} if

there is an accepting run of \mathcal{A} on w . The (unweighted) *language* of \mathcal{A} is $L(\mathcal{A}) = \{w : w \text{ is accepted by } \mathcal{A}\}$. For $q \in Q$, we denote by \mathcal{A}^q the automaton \mathcal{A} with the single initial state q . The cost of an accepting run is the sum of the weights of the transitions that constitute the run³. Formally, let $r = r_0 r_1 \dots r_n$ be an accepting run of \mathcal{A} on w , and let $d = d_1 \dots d_n \in \Delta^*$ be the corresponding sequence of transitions. The cost of r is $\text{cost}(\mathcal{A}, r) = \sum_{i=1}^n c(d_i)$. The cost of w , denoted $\text{cost}(\mathcal{A}, w)$, is the minimal cost over all accepting runs of \mathcal{A} on w . Thus, $\text{cost}(\mathcal{A}, w) = \min\{\text{cost}(\mathcal{A}, r) : r \text{ is an accepting run of } \mathcal{A} \text{ on } w\}$. For completeness, if $w \notin L(\mathcal{A})$ we set $\text{cost}(\mathcal{A}, w) = \infty$.

For two WFAs \mathcal{A}_1 and \mathcal{A}_2 , and $\alpha \geq 1$, we say that \mathcal{A}_1 α -*approximates* \mathcal{A}_2 if $L(\mathcal{A}_1) = L(\mathcal{A}_2)$ and for all words $w \in \Sigma^*$, we have $\text{cost}(\mathcal{A}_1, w) \leq \alpha \cdot \text{cost}(\mathcal{A}_2, w)$. When both \mathcal{A}_1 1-approximates \mathcal{A}_2 and \mathcal{A}_2 1-approximates \mathcal{A}_1 , we say that \mathcal{A}_1 and \mathcal{A}_2 are *equivalent*.

2.2 Online algorithms A *problem* associates with each possible input I a set $F(I)$ of feasible solutions. In an *optimization problem* (of cost minimization), each solution in $F(I)$ has a cost in $\mathbb{R}^{\geq 0}$, and the goal is to find a feasible solution that minimizes the cost.

An *online algorithm* for an optimization problem P is an algorithm that gets as input a finite sequence of requests, and has to process each request (and end up in a feasible solution) without knowing the requests yet to come. In contrast, an *offline algorithm* for P gets the entire sequence in advance, and its decisions as to how to process a request may depend on the requests yet to come.

Formally, if we denote by Σ the set of requests, and denote by A the set of actions that are available to the algorithm, then an online algorithm corresponds to a function $g : \Sigma^+ \rightarrow A$. The processing of an input sequence $\sigma_1 \dots \sigma_n$ by g is then $g(\sigma_1)$, $g(\sigma_1 \sigma_2)$, $g(\sigma_1 \sigma_2 \sigma_3)$, \dots . In typical optimization problems, there is a cost function *action_cost* : $A \rightarrow \mathbb{R}^{\geq 0}$ that associates a cost with each action. The cost of processing an input sequence is the sum of the costs of the actions taken in order to process it. The performance of an online algorithm is typically worse than that of an offline

³In general, a WFA may be defined with respect to any semiring $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$. The cost of a run is then the semiring product of the weights along it, and the cost of an accepted word is the semiring sum over all accepting runs on it. For the modeling of online algorithms, we focus on weighted automata defined with respect to the *min-sum semiring*, $(\mathbb{R}^{\geq 0} \cup \{\infty\}, \min, +, \infty, 0)$ (sometimes called the *tropical semiring*), as defined above. Also, some work assigns costs also to initial and accepting states. We do not need such costs for the modeling of online algorithms, and work with a definition that omits them.

algorithm for the same problem. For analyzing the performance of online algorithms we use *competitive analysis*, which compares the two performances.

For an online algorithm g and an input $w \in \Sigma^+$, let $g(w)$ denote the cost of processing w by g , and let $\text{OPT}(w)$ denote the cost of processing w by the optimal offline algorithm. We say that an online algorithm g is α -*competitive* if there exists a constant β such that for all input sequences $w \in \Sigma^+$ we have that $g(w) \leq \alpha \cdot \text{OPT}(w) + \beta$. The *competitive ratio* of g is the smallest α for which g is α -competitive. In the rest of the paper we restrict attention to the multiplicative factor α and ignore β . As we show in the full version, all our results can be easily extended to handle the additive factor β .

Our analysis of online algorithms takes into account the extra memory that the online algorithm may require in order to compete with the offline algorithm. Formally, we have the following.

DEFINITION 2.1. *For a set S of configurations, a competitive ratio $\alpha \geq 1$, and an integer $r \geq 0$, we say that an optimization problem P has competitive ratio (α, r) with memory S , if there is an online algorithm g for P that uses an extension of the memory S by r Boolean variables, and g is α -competitive with respect to an optimal offline algorithm that uses memory S .*

3 An Automata-Theoretic Approach to Reasoning about Online Algorithms

In this section we describe an automata-theoretic approach to reasoning about online algorithms. We first characterize optimization problems for which the approach can be applied, and argue that typical optimization problems satisfy our characterization. We then describe how, by modeling optimization problems by weighted nondeterministic automata, we can reduce reasoning about the competitive ratio and the memory required by online algorithms, to reasoning about determination of such automata.

3.1 Finite-state online algorithms Recall that an online algorithm corresponds to a function $g : \Sigma^+ \rightarrow A$ that maps sequences of requests (the history of the interaction so far) to an action to be taken. In general, the algorithm induces an infinite state space, as it may be in different states after processing different input sequences in Σ^* . Indeed, modeling of online algorithms by request-answer games gives rise to games with infinitely many positions [3]. For a finite set S of configurations, we say that g *uses memory S* , if there is a regular mapping of Σ^* into S such that g behaves in the same manner on words that are mapped to the

same configuration.

We model the set of online algorithms that use memory S and solve an optimization problem P with requests in Σ and actions in A , by a WFA $\mathcal{A}_P = \langle \Sigma, S, \Delta, c, S_0, S \rangle$, such that Δ and c describe transitions between configurations and their costs, and S_0 is a set of possible initial configurations. Formally, $\Delta(s, \sigma, s')$ if the set $A' \subseteq A$ of actions that process the request σ from configuration s by updating the configuration to s' is non-empty, in which case $c(\langle s, \sigma, s' \rangle) = \min_{a \in A'} \text{action_cost}(a)$. Note that all the states of \mathcal{A}_P are accepting. Thus, \mathcal{A}_P assigns a cost to all sequences in Σ^* .

Many optimization problems have online algorithms that require finite memory, or have finite memory variants that are obtained by imposing natural bounds. We give a few examples below. In the full version, we describe in addition the WFA corresponding to the load-balancing problem.

3.1.1 Paging [21] In the *paging* problem we have a two-level memory hierarchy: A slow memory that contains n different pages, and a *cache* that contains at most k different pages (typically, $k \ll n$). Pages that are in the cache can be accessed at zero cost. If a request is made to access a page that is not in the cache, the page should be brought into the cache, at a cost of 1, and if the cache is full, some other page should first be evicted from the cache. The paging problem is, given a sequence of requested pages, to decide which page to evict whenever an eviction is needed. The goal is to minimize the total cost.

A paging problem P with parameters n and k induces a WFA $\mathcal{A}_P = \langle \Sigma, S, \Delta, c, S_0, S \rangle$, where $\Sigma = \{1, \dots, n\}$ is the set of possible requests (page indices), $S = \{C \subseteq \{1, \dots, n\} : |C| \leq k\}$ is a set of finite configurations, each describing the set of pages currently in the cache, Δ and c describe how (and at which cost) requests are served, and $S_0 = \{\emptyset\}$, indicating that the cache is initially empty. Thus, $\Delta(C, i, C')$ iff one of the following holds: (1) $i \in C$, in which case $C' = C$ and $c(\langle C, i, C' \rangle) = 0$, (2) $i \notin C$, $|C| < k$, and $C' = C \cup \{i\}$, in which case $c(\langle C, i, C' \rangle) = 1$, or (3) $i \notin C$, $|C| = k$, and there is $j \in C$ such that $C' = (C \setminus \{j\}) \cup \{i\}$, in which case $c(\langle C, i, C' \rangle) = 1$. Note that by the definition of S , a configuration stores only the set of pages currently in the cache, and there are no provisions for storing any extra information such as time-stamps, etc. A different automaton for the problem could have defined S in a way that allows the storage of such extra information. We will elaborate on this point in the sequel.

3.1.2 The k -server problem [19] The paging problem can be viewed as a special case of the k -server problem. There, we have k servers in a metric space $M = \langle V, d \rangle$, where V is a set of points and $d : V \times V \rightarrow \mathbb{R}^{\geq 0}$ is a distance function. The input to the problem is a sequence of points, each point should be served by moving a server to it (if no server is there), and the goal is to minimize the sum of distances that the servers move.

A k -server problem P with parameters k and $M = \langle V, d \rangle$, for a finite set V , induces a WFA $\mathcal{A}_P = \langle V, V^k, \Delta, c, \{s_0\}, V^k \rangle$, where each state corresponds to a configuration of the servers (for simplicity, we allow several servers to cover the same point), Δ and c describe how (and at which cost) servers may move, and s_0 is an initial configuration defined by the problem. Thus, $\Delta(s, v, s')$ iff one of the following holds: (1) there is $1 \leq j \leq k$ such that $v = s(j)$ and $s' = s$, in which case $c(\langle s, v, s' \rangle) = 0$, or (2) $s(j) \neq v$ for all $1 \leq j \leq k$, there is $1 \leq j \leq k$ such that $v = s'(j)$ and for all $l \neq j$, we have $s'(l) = s(l)$, in which case $c(\langle s, v, s' \rangle) = d(s(j), s'(j))$.⁴

3.1.3 The ski-rental problem In the ski-rental problem someone goes on a ski vacation whose length is not known in advance. Each morning he has to decide between renting skis (\$1 per day) and buying skis (\$ y). The goal is to minimize the expense. Here, making the problem finite-state requires the introduction of a finite bound M on the length of the vacation. Note that since M may be bigger than y , the challenge of an algorithm that knows M and does not know the length of the vacation in advance is similar to the challenge of an algorithm that does not know M . Indeed, studies of the problem usually refer to its finite-leasing version, in which the bound M is part of the input [4]. A ski-rental problem P with parameters y and M induces a WFA $\mathcal{A}_P = \langle \{a\}, \{0, \dots, M+1\}, \Delta, c, \{0\}, \{0, \dots, M+1\} \rangle$, where $\Delta(s, a, s')$ iff (1) $0 \leq s < M$ and $s' = s+1$, in which case $c(\langle s, a, s' \rangle) = 1$, (2) $0 \leq s < M$ and $s' = M+1$, in which case $c(\langle s, a, s' \rangle) = y$, or (3) $s = s' = M+1$, in which case $c(\langle s, a, s' \rangle) = 0$. Note that the alphabet of \mathcal{A}_P is a singleton letter, as we only care whether the vacation ends (the input word ends too) or not (the next letter is read).

3.1.4 Δ -paid exchange static list accessing [4] In this problem we have a static (fixed) linked list of n items. Each request is for an element of the list to be accessed. A request to access the i -th element in the list

necessitates the traversal of i links, which costs i . After servicing a request, the list may be rearranged in the hope of better servicing future requests. Rearranging the list can be done by a series of exchanges of two consecutive items. Each exchange costs $\Delta > 1$.

While attempts to model the problem with *metric task systems* fail [4], it is not hard to see that P with parameters n and Δ induces a WFA $\mathcal{A}_P = \langle \Sigma, S, S \times \Sigma \times S, c, S_0, S \rangle$, where $\Sigma = \{1, \dots, n\}$ and S is the set of all $n!$ permutations of $\{1, \dots, n\}$, representing all the possible arrangements of the elements in the list. The cost of a transition $\langle s, i, s' \rangle$ is $j + \Delta k$, where j is the position of i in the permutation s , and k is the minimal number of exchanges needed to transform the list from the ordering s to the ordering s' .

We note that while the size of \mathcal{A}_P is bounded by $|S|^2 \cdot |\Sigma|$, its computation may be complex, as demonstrated by the list accessing example. Note, however, that the source of the complexity is the fact that we compressed all the internal steps of the algorithm into one transition. Instead, one can enrich the alphabet of \mathcal{A}_P and encode each request as a sequence of letters, thus allowing \mathcal{A}_P to process each request by a series of internal steps, and avoiding such compressions.

3.2 Relating online algorithms and determinization by pruning In this section we reduce problems concerning online algorithms to questions about weighted automata. We first need some definitions. For two WFAs $\mathcal{A} = \langle \Sigma, Q, \Delta, c, Q_0, F \rangle$ and $\mathcal{A}' = \langle \Sigma, Q', \Delta', c', Q'_0, F \rangle$, we say that \mathcal{A} *embodies* \mathcal{A}' if $Q'_0 \subseteq Q_0$, $\Delta' \subseteq \Delta$, and c' agrees with c on Δ' . Thus, \mathcal{A}' can be obtained from \mathcal{A} by decreasing its nondeterminism. For a WFA $\mathcal{A} = \langle \Sigma, Q, \Delta, c, Q_0, F \rangle$ and an integer $r \geq 0$, the r -*refinement* of \mathcal{A} is the WFA \mathcal{A}_r obtained by refining the state space of \mathcal{A} by r Boolean variables. Formally, $\mathcal{A}_r = \langle \Sigma, Q \times 2^{\{1, \dots, r\}}, \Delta_r, c_r, Q_0 \times 2^{\{1, \dots, r\}}, F \times 2^{\{1, \dots, r\}} \rangle$, where each state $\langle q, f \rangle \in Q \times 2^{\{1, \dots, r\}}$ maintains, in addition to the state q of \mathcal{A} , also a subset f of $\{1, \dots, r\}$, corresponding to a truth assignment for the new variables. The transition relation Δ_r and the cost function c_r are the expected extensions of Δ and c . That is, for every $f, f' \in 2^{\{1, \dots, r\}}$, we have that $\Delta_r(\langle q, f \rangle, a, \langle q', f' \rangle)$ iff $\Delta(q, a, q')$, in which case $c_r(\langle \langle q, f \rangle, a, \langle q', f' \rangle \rangle) = c(\langle q, a, q' \rangle)$. Thus, each state of \mathcal{A} has 2^r isomorphic copies in \mathcal{A}_r .

DEFINITION 3.1. *Consider a WFA \mathcal{A} , an approximation factor $\alpha \geq 1$, and an integer $r \geq 0$. We say that \mathcal{A} is (α, r) -determinizable by pruning ((α, r) -DBP, for short) if the r -refinement of \mathcal{A} embodies a DWFA that α -approximates \mathcal{A} .*

⁴Note that both in the paging problem and here, we restrict attention to *lazy* algorithms, which minimize the change of configurations so that only the current request is served. By [19], for every non-lazy algorithm, there exists a lazy one that performs at least as well.

Note that when $\alpha = 1$, the embodied DWFA is equivalent to \mathcal{A} . Also, when $r = 0$, no refinement takes place, and the embodied automaton has the same state space as \mathcal{A} . When \mathcal{A} is (1,0)-DBP, we say that \mathcal{A} is DBP.

Let P be an optimization problem, and let $\mathcal{A}_P = \langle \Sigma, S, \Delta, c, S_0, S \rangle$ be a WFA for its algorithms that use memory S . Given a finite sequence of requests $w \in \Sigma^*$, each run of \mathcal{A}_P on w corresponds to a way of serving the requests in w by an algorithm with configurations in S . The set of all runs include all such algorithms, thus the cost of w in \mathcal{A}_P is the cost of w in an optimal offline algorithm that uses memory S . Indeed, the semantics of WFA over the tropical semiring, in which the cost of a word is the minimum cost of some run on it, guarantees that the cost would be calculated according to the best guess. On the other hand, an online algorithm has to process each request as soon as it arrives, without knowing the requests yet to arrive. Accordingly, an online algorithm that uses memory S corresponds to a DWFA embodied in \mathcal{A}_P . Indeed, for every configuration $s \in S$ of the problem and request $\sigma \in \Sigma$, the algorithm suggests a particular way to process σ from s , inducing a particular transition $\langle s, \sigma, s' \rangle \in \Delta$. Moreover, a refinement of \mathcal{A}_P maintains the correspondence between its transitions and the actions of the algorithms (note that this correspondence is lost if we consider unrestricted determinization of \mathcal{A}_P). Hence, a DWFA embodied in a refinement of \mathcal{A}_P corresponds to an online algorithm with an extended memory. Formally, we have the following.

THEOREM 3.1. *Consider an optimization problem P and a set S of configurations. Let \mathcal{A}_P be a WFA with state space S that models online algorithms for P that use memory S . For all $\alpha \geq 1$ and $r \geq 0$, the problem P has competitive ratio (α, r) with memory S iff \mathcal{A}_P is (α, r) -DBP.*

4 Determinization and Approximated Determinization by Pruning

In this section we study the problem of determinization by pruning. We show that deciding whether a given WFA is DBP (the *DBP problem*, for short) can be done in polynomial time. On the other hand, deciding whether a given WFA is $(\alpha, 0)$ -DBP, for $\alpha > 1$ (the *approximated DBP problem*, for short) is NP-complete. In both cases, when the answer is positive, returning a witness DWFA requires no extra cost.

4.1 Deciding determinization by pruning The polynomial-time algorithm for the DBP problem is our most challenging technical result. For clarity, we

first describe a polynomial-time algorithm for deciding whether a given NFA (that is, a WFA with no costs) is DBP (that is, embodies an equivalent DFA).

THEOREM 4.1. *The DBP problem for NFA can be solved in polynomial time.*

Proof. Consider an NFA $\mathcal{A} = \langle \Sigma, Q, \Delta, Q_0, F \rangle$. We inductively define a sequence $H_0, H_1, \dots \subseteq Q \times Q$ of relations as follows.

$$H_0 = (F \times F) \cup ((Q \setminus F) \times Q), \text{ and for } i > 0 : \\ H_i = H_{i-1} \cap \{ \langle q, q' \rangle : \text{for all } a \in \Sigma \text{ there exists} \\ v' \in \delta(q', a) \text{ such that for all } v \in \delta(q, a) \\ \text{we have } H_{i-1}(v, v') \}.$$

Intuitively, $H_i(q, q')$ means that there is a DFA \mathcal{A}' embodied in \mathcal{A} such that all the words of length at most i accepted from q in \mathcal{A} are also accepted from q' in \mathcal{A}' . Since $H_0 \supseteq H_1 \supseteq H_2 \supseteq \dots$, the sequence of relations eventually reaches a fixed-point, which we denote by H . For two states q and q' , we say that q' covers q if $H(q, q')$. The relation H induces an NFA $\mathcal{A}^H = \langle \Sigma, Q, \Delta^H, Q_0^H, F \rangle$ embodied in \mathcal{A} , where $q_0 \in Q_0^H$ iff $q_0 \in Q_0$ and q_0 covers all the states in Q_0 , and for every $q, v \in Q$ and $a \in \Sigma$, we have that $\Delta^H(q, a, v)$ iff $\Delta(q, a, v)$ and v covers all the states in $\delta(q, a)$. Note that the set Q_0^H may be empty, and that for some q and a it may be that $\delta^H(q, a) = \emptyset$ even though $\delta(q, a) \neq \emptyset$. In the full version we show that $Q_0^H \neq \emptyset$ iff \mathcal{A} is DBP. Since the calculation of H can be done in polynomial time, we are done.

Note that, like the algorithm for DFA minimization, our algorithm calculates a fixed-point over pairs of states. The fixed-point here, however, is different and more complicated, as it involves a universal requirement nested inside an existential requirement. We found the result to be quite surprising. Indeed, in the full version we show that a slightly different decision problem, in which the embodied automaton is deterministic modulo initial nondeterminism (that is, $|\delta(q, a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$, yet Q_0 may not be a singleton) is NP-complete.

We now move to the DBP problem for WFA. Like the algorithm in the unweighted case, the polynomial algorithm we present below is based on a fixed-point calculation, that for each pair $\langle q, q' \rangle$ of states of \mathcal{A} , compares the behavior of embodied deterministic automata with initial state q' to the behavior of the nondeterministic automaton \mathcal{A}^q , over words of increasing length. The setting here, however, is much more difficult. First, the characterization associated with each pair is not Boolean: it is not enough to remember whether one

can deterministically accept from q' the same words as from q — the characterization has to further refine this information and refer to the possibly different costs involved. Second, while in the unweighted setting it is clear that the calculation would reach a fixed-point in a polynomial number of steps, in the weighted setting it may well be that as the length of the words considered increases, so does the cost difference, and it is not clear how to force the calculation to reach a fixed point.

THEOREM 4.2. *The DBP problem for WFA can be solved in polynomial time.*

Proof. We first need some notations. For every $r \in \mathbb{R}$ we have $-\infty < r < \infty$, and we allow expressions of the form $\infty \pm r$, $-\infty \pm r$, $\infty + \infty$, and $(-\infty) + (-\infty)$, with the usual meaning. For every $i \geq 0$, let $\Sigma^{\leq i} = \{w \in \Sigma^* : |w| \leq i\}$. Let \mathcal{A} and \mathcal{A}' be two WFAs over the same alphabet Σ . Given a subset $S \subseteq \Sigma^*$, we define the cost difference between \mathcal{A} and \mathcal{A}' over S to be $\text{costdiff}(\mathcal{A}', \mathcal{A}, S) = \sup_{w \in S \cap L(\mathcal{A})} [\text{cost}(\mathcal{A}', w) - \text{cost}(\mathcal{A}, w)]$. Note that if $S \cap L(\mathcal{A}) = \emptyset$ then $\text{costdiff}(\mathcal{A}', \mathcal{A}, S) = -\infty$, and that if there is a word $w \in S \cap L(\mathcal{A}) \setminus L(\mathcal{A}')$ then $\text{costdiff}(\mathcal{A}', \mathcal{A}, S) = \infty$. Also note that, unless both WFAs accept the empty language, \mathcal{A} and \mathcal{A}' are equivalent iff $\text{costdiff}(\mathcal{A}', \mathcal{A}, \Sigma^*) = 0$ and $\text{costdiff}(\mathcal{A}, \mathcal{A}', \Sigma^*) = 0$.

Consider a WFA $\mathcal{A} = \langle \Sigma, Q, \Delta, c, Q_0, F \rangle$. Let $n = |Q|$. Our algorithm calculates a sequence of functions $f_0, f_1, \dots, f_{2n^2-1} : Q \times Q \rightarrow \mathbb{R} \cup \{-\infty, \infty\}$, such that the following holds.

- For $0 \leq i \leq n^2 - 1$, the function $f_i(q, q')$ measures how well the state q' can deterministically simulate the state q , over words of length at most i . Formally, for every embodied deterministic automaton \mathcal{A}' equivalent to \mathcal{A} (if exists), and every pair of states $q, q' \in Q$ such that q' is reachable in \mathcal{A}' , we have that $f_i(q, q') = \text{costdiff}(\mathcal{A}'^{q'}, \mathcal{A}^q, \Sigma^{\leq i})$. The value $-\infty$ is assigned to $f_i(q, q')$ when there are no words in $\Sigma^{\leq i}$ that can be accepted from q , and the value ∞ is assigned when there is a word that can be accepted from q but not from q' .
- For $n^2 \leq i \leq 2n^2 - 1$, the function $f_i(q, q')$ is similar, only that it takes cycles into account, and maps to ∞ pairs for which the cost difference has not stabilized yet, which indicates that it cannot be bounded.

The sequence of functions $f_0, f_1, \dots, f_{2n^2-1}$ is defined as follows.

- At initialization, $f_0(q, q')$ is $-\infty$ if $q \notin F$, is 0 if $q \in F$ and $q' \in F$, and is ∞ if $q \in F$ and $q' \notin F$.

- For $1 \leq i \leq n^2 - 1$, we have $f_i(q, q') = \max\{f_{i-1}(q, q'), \max_{a \in \Sigma} f_i(q, q', a)\}$.
- For $n^2 \leq i \leq 2n^2 - 1$, we have $f_i(q, q') = \infty$ if $\max_{a \in \Sigma} f_i(q, q', a) > f_{i-1}(q, q')$, and $f_i(q, q') = f_{i-1}(q, q')$ otherwise.

In the above, for every $1 \leq i \leq 2n^2 - 1$ and $a \in \Sigma$, the function $f_i(q, q', a)$ is defined as follows.

$$f_i(q, q', a) = \min_{u' \in \rho_{i-1}(q', a)} \max_{u \in \delta(q, a)} [f_{i-1}(u, u') + c(q', a, u') - c(q, a, u)],$$

where the set $\rho_0(q', a) = \delta(q', a)$, and for $1 \leq i \leq 2n^2 - 1$, we have

$$\rho_i(q', a) = \{u' \in \rho_{i-1}(q', a) : \max_{u \in \delta(q', a)} [f_{i-1}(u, u') + c(q', a, u') - c(q', a, u)] \leq 0\}.$$

In the expression above for $f_i(q, q', a)$, in case that for all $u \in \delta(q, a)$ we have that $L(\mathcal{A}^u) \cap \Sigma^{\leq i-1} = \emptyset$, we set $f_i(q, q', a) = -\infty$ (note that this also covers the case $\delta(q, a) = \emptyset$). In case there is $u \in \delta(q, a)$ such that $L(\mathcal{A}^u) \cap \Sigma^{\leq i-1} \neq \emptyset$ and $\rho_{i-1}(q', a) = \emptyset$, we set $f_i(q, q', a) = \infty$.

Intuitively, a state $u' \in \rho_i(q', a)$ is a “witness” to the fact that $f_i(q', q', a) \leq 0$, i.e., to the fact that q' can deterministically simulate itself over words in $\Sigma^{\leq i}$ that start with a . Clearly, only such witnesses can be a -successors of q' in an embodied DWFA that is equivalent to \mathcal{A} .

We argue that the sequence of functions reaches a fixed-point in some iteration $1 \leq j \leq 2n^2 - 1$, and that \mathcal{A} is DBP iff there is a state $q'_0 \in Q_0$ such that for every $q_0 \in Q_0$, we have that $f_j(q_0, q'_0) \leq 0$. Also, in case \mathcal{A} is DBP, then every DWFA that consists of transitions that use the witnesses from the last iteration (that is, whose transition relation assigns successors according to ρ_j) is equivalent to \mathcal{A} . A detailed proof can be found in the full version.

4.2 Deciding approximated determinization by pruning We now turn to the approximated-DBP problem, and show that it is much harder. We first study the problem of approximation of a WFA by a given embodied DWFA.

LEMMA 4.1. *Consider a WFA \mathcal{A} , an embodied DWFA \mathcal{A}' , and an approximation factor $\alpha \geq 1$. Deciding whether \mathcal{A}' α -approximates \mathcal{A} can be done in polynomial time.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, \Delta, c, Q_0, F \rangle$ and $\mathcal{A}' = \langle \Sigma, Q', \Delta', c', Q'_0, F' \rangle$. Consider first the case $\alpha = 1$.

Then, the algorithm is similar to the one used for checking whether \mathcal{A} is DBP, only that now \mathcal{A}' is given. Accordingly, the functions f_i are defined for pairs in $Q \times Q'$, and when calculating $f_i(q, q', a)$, we only have to consider the given (if any) a -successor of q' in \mathcal{A}' , instead of the set $\rho_{i-1}(q', a)$.

Now, given $\alpha > 1$, we further modify the algorithm by scaling all the edges of \mathcal{A} by α . More formally, we define a sequence of functions $g_0, g_1, \dots, g_{2n^2-1} : Q \times Q' \rightarrow \mathbb{R} \cup \{-\infty, \infty\}$ that is similar to the sequence f_i , except that for every $1 \leq i \leq 2n^2 - 1$ and $a \in \Sigma$, we have $g_i(q, q', a) = \max_{u \in \delta(q, a)} [g_{i-1}(u, \delta'(q', a)) + c(q', a, \delta'(q', a))] - \alpha \cdot c(q, a, u)$.

Before we use Lemma 4.1 for solving the approximated DBP problem, we note its application in reasoning about online algorithms. Indeed, by Theorem 3.1, we have the following.

COROLLARY 4.1. *Consider an optimization problem P and a finite set S of configurations. Given an online algorithm g with memory S , and a competitive ratio $\alpha \geq 1$, the problem of deciding whether g is α -competitive with respect to an offline algorithm with memory S can be solved in polynomial time.*

In light of Lemma 4.1, one may be tempted to believe that by using the same ideas one can extend Theorem 4.2 to handle an approximation factor $\alpha > 1$. Unfortunately, as the next theorem shows, unless $P=NP$, this can not be the case. Essentially, the property that if $u' \in \rho_{i-1}(q', a)$ then u' is such that for every $q \in Q$ the minimum in the expression for $f_i(q, q', a)$ is achieved with u' , which is crucial to proving Theorem 4.2, is no longer true when $\alpha > 1$.

THEOREM 4.3. *The approximated-DBP problem is NP-complete.*

Proof. Membership in NP follows from Lemma 4.1. We prove NP-hardness by a reduction from 3-SAT. Given $\alpha > 1$ and a 3-SAT formula $\psi = \bigwedge_{j=1}^m C_j$ over the variables x_1, \dots, x_n , we build a WFA \mathcal{A} that is $(\alpha, 0)$ -DBP iff ψ is satisfiable. We assume without loss of generality that no clause in ψ contains both a variable and its negation. The alphabet of \mathcal{A} is $\Sigma = \{a\} \cup \{C_1, \dots, C_m\}$, and \mathcal{A} is given in Figure 1.

For every $1 \leq i \leq n$ and every literal $l_i \in \{x_i, \neg x_i\}$, the edge between l_i and p_i (which for lack of space is unlabeled in the figure) stands for m transitions, one for each of the letters C_1, \dots, C_m , and the cost of a transition $\langle l_i, C_j, p_i \rangle$ is α if the literal $\neg l_i$ appears in the clause C_j , and is 1 otherwise. Recall that no clause in ψ contains both a variable and its negation. Hence, for every $1 \leq i \leq n$ and every $1 \leq j \leq m$, at least one of

the transitions $\langle x_i, C_j, p_i \rangle$, and $\langle \neg x_i, C_j, p_i \rangle$ costs 1. It follows that \mathcal{A} with initial state p_0 accepts exactly all words of the form $(a \cdot (C_1 + \dots + C_m))^n$ with cost n . In addition, \mathcal{A} has m components such that for every $1 \leq j \leq m$, the DWFA \mathcal{A} with initial state q_0^j accepts the word $(aC_j)^n$ with a lower cost (recall that $\alpha > 1$) of $(n-2)/\alpha + 2$. In the full version we show that \mathcal{A} is $(\alpha, 0)$ -DBP iff ψ is satisfiable.

We note that an adjustment to the costs of the transitions of the WFA used in the proof of Theorem 4.3 shows that the approximated-DBP problem is NP-hard already for an additive approximation factor (that is, when the embodied DWFA \mathcal{A}' is such that there is $\beta \geq 0$ such that for all $w \in \Sigma^*$, we have $\text{cost}(\mathcal{A}', w) \leq \beta + \text{cost}(\mathcal{A}, w)$).

By Theorem 3.1 (and the fact that every WFA induces an online algorithm), we can conclude with the following.

COROLLARY 4.2. *Consider an optimization problem P and a finite set S of configurations. The problem of deciding whether P has competitive ratio $(\alpha, 0)$ with memory S can be solved in polynomial time for $\alpha = 1$, and is NP-complete for $\alpha > 1$.*

5 Determinization and Approximated Determinization by Refinement and Pruning

In this section we study the problem of determinization by refinement and pruning. We first show that extension of the memory is hopeless in an effort to be as good as an optimal offline algorithm.

THEOREM 5.1. *For all integers $r \geq 0$, a WFA \mathcal{A} is $(1, r)$ -DBP iff it is $(1, 0)$ -DBP.*

Proof. Clearly, if \mathcal{A} is $(1, 0)$ -DBP, then it is also $(1, r)$ -DBP. We prove that if \mathcal{A} is $(1, r)$ -DBP for some $r \geq 0$, then it is also $(1, 0)$ -DBP. For a refinement \mathcal{A}_r of \mathcal{A} , we say that a DWFA \mathcal{D}_r , obtained by pruning \mathcal{A}_r , is *simple* if for each state q of \mathcal{A} there is at most one subset $f \subseteq \{1, \dots, r\}$ such that the state $\langle q, f \rangle$ is reachable in \mathcal{D}_r . If \mathcal{A}_r can be pruned to a simple equivalent \mathcal{D}_r , then by omitting the $2^{\{1, \dots, r\}}$ element of each state we get an equivalent deterministic pruning of \mathcal{A} , and we are done.

Assume now that no simple equivalent pruning exists; i.e., in every equivalent DWFA \mathcal{D}_r that is obtained by pruning \mathcal{A}_r , there exists a state q and two subsets $f_1, f_2 \in 2^{\{1, \dots, r\}}$, such that both $\langle q, f_1 \rangle$ and $\langle q, f_2 \rangle$ are reachable. Then, there must be a word t_1 , accepted from $\langle q, f_1 \rangle$ with a certain cost and from $\langle q, f_2 \rangle$ with a higher cost (or not at all). Let h_2 be a word that allows \mathcal{D}_r to reach $\langle q, f_2 \rangle$. It follows that either the word $h_2 \cdot t_1$ is not accepted by \mathcal{D}_r , or it is accepted with

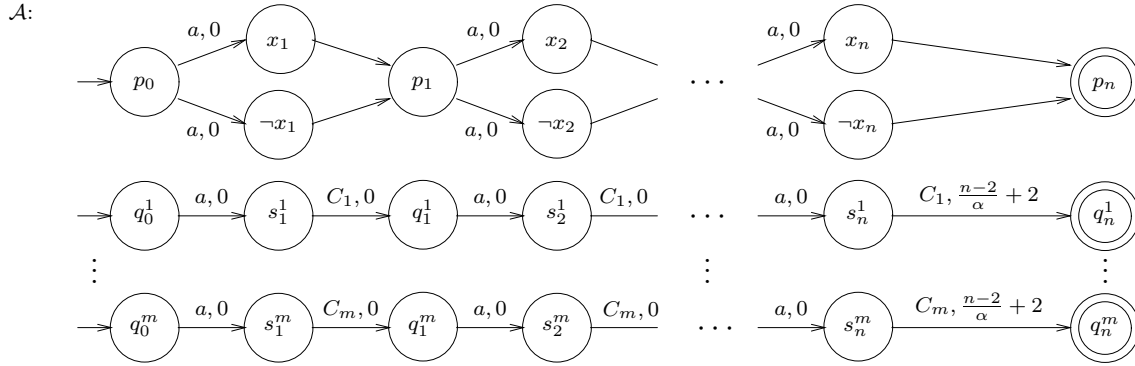


Figure 1: A WFA for a 3-SAT formula.

a cost higher than that with which it is accepted by \mathcal{A} , and we have reached a contradiction.

On the other hand, an extension of the memory may help in achieving a better competitive ratio:

THEOREM 5.2. *For all $\alpha > 1$ and $n \geq 1$, there exists a WFA \mathcal{A} that is (α, n) -DBP but not $(\alpha, n - 1)$ -DBP.*

Proof. The WFA \mathcal{A} appearing in Figure 2 is $(2, 1)$ -DBP but not $(2, 0)$ -DBP. Note that the language of \mathcal{A} consists of words of the form $x \cdot a \cdot y$, for $x, y \in \{a, b\}$. The cost of an accepted word is 1 if $x = y$ and 2 otherwise. In the refined 2-DBP, the cost of an accepted word is 2 if $x = y$ and 4 otherwise. The automaton \mathcal{A} can be generalized for any $n, \alpha > 1$ to an automaton $\mathcal{A}_{n, \alpha}$ of size $O(n \cdot 2^n)$ with a maximal branching degree of 2, whose language consists of all the words of the form $w \cdot a \cdot v$ where $w, v \in \{a, b\}^n$, such that the cost of an accepted word is 1 if $w = v$ and is α otherwise, and that $\mathcal{A}_{n, \alpha}$ is (α, n) -DBP but not $(\alpha, n - 1)$ -DBP.

Theorem 5.2 also follows from specific examples studied in the literature, showing that online algorithms that can store additional information can achieve better competitive ratios (for example, [6] shows a lower bound of $23/11$ on the competitiveness of any deterministic trackless online algorithm for the 2-server problem⁵; whereas [10] shows that the competitive ratio of the Work Function Algorithm, which is also deterministic, but not trackless, for the 2-server problem is 2). Nonetheless, the proof of Theorem 5.2 serves to pinpoint the source of this phenomenon.

⁵Being memoryless is a stronger restriction than being trackless. Thus, this lower bound is valid also for memoryless algorithms.

6 Discussion

The automata-theoretic approach we have described involves an explicit representation of the set S of configurations. One of the main challenges in formal verification is the need to cope with very big, often infinite, state spaces. *Symbolic reasoning* [7] is a leading approach for doing so. There, S is given symbolically (say, by a characteristic function), and the operations allowed to the verification algorithm are symbolic too. Since our algorithms are based on a fixed-point computation of a set of relations or functions, which are typically amenable to symbolic implementation, we are optimistic about adjusting them to the symbolic setting. Another challenge in our setting is that we would like to prove general properties of an online algorithm, rather than properties of instances corresponding to given parameters. This challenge is addressed in formal verification by means of *parametric reasoning* [13]. There, we reason about a system with many identical processes by studying properties of one of the processes. Parametric reasoning is, in general, undecidable. However, in the last decade there has been extensive research aimed at finding settings for which the problem is decidable, and on developing methods that are sound but incomplete. We are now examining their application to the setting of online algorithms. It is important to note that the field of formal verification has a history of successful implementations of algorithms with seemingly-infeasible complexity. For example, the tool MONA successfully decides the satisfiability of monadic second-order logic formulas – a problem whose complexity is non-elementary [12].

Finally, while we are able to decide whether a given online algorithm g has a given competitive ratio (Corollary 4.2), we left open the problem of finding the competitive ratio of g . Clearly, finding a finite upper bound on the competitive ratio would enable us to apply

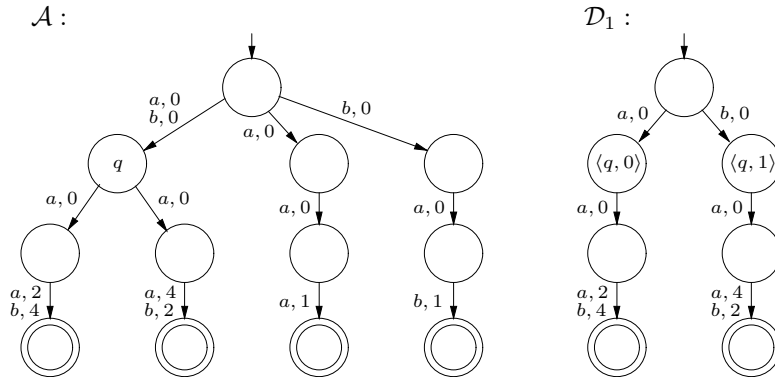


Figure 2: A WFA \mathcal{A} and its refined 2-determinization by pruning \mathcal{D}_1 .

Corollary 4.2 and search for it. In the WFA formalism, this is reduced to finding, given a WFA \mathcal{A} , a finite bound γ such that \mathcal{A} is $(\gamma, 0)$ -DBP (or deciding that no such γ exists). We believe that such a bound can be found by analyzing the cost of cycles of \mathcal{A} , and we leave open the problem of doing it in polynomial time.

Acknowledgements We thank Marek Chrobak, Lawrence Larmore, and Nati Linial for helpful discussions.

References

- [1] A.V. Aho, P.J. Denning, and J.D. Ullman. Principles of optimal page replacement. *Journal of the ACM*, 18(1):80–93, 1971.
- [2] T. Ball, B. Cook, V. Levin, and S.K. Rajamani. Slam and static driver verifier: Technology transfer of formal methods inside microsoft. In *Integrated Formal Methods*, pages 1–20, 2004.
- [3] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *algorithmica*, 11(2):2–14, 1994.
- [4] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge Univ. Press, 1998.
- [5] A. Borodin, N. Linial, and M.E. Saks. An optimal online algorithm for metrical task systems. In *Proc. 19th STOC*, pages 373–382, 1987.
- [6] W.W. Bein and L.L. Larmore. Trackless Online Algorithms for the Server Problem. *Information Processing Letters*, 74:73–79, 2000.
- [7] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
- [8] A. Chakrabarti, K. Chatterjee, T.A. Henzinger, O. Kupferman, and R. Majumdar. Verifying quantitative properties using bound functions. In *Proc. 13th CHARME*, LNCS 3725, pages 50–64. Springer, 2005.
- [9] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
- [10] M. Chrobak and L.L. Larmore. The server problem and on-line games. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:11–64, 1991.
- [11] M. Chrobak and L.L. Larmore. Private communication. 2008.
- [12] J. Elgaard, N. Klarlund, and A. Möller. Mona 1.x: new techniques for WS1S and WS2S. In *Proc. 10th CAV*, LNCS 1427, pages 516–520, 1998.
- [13] E.A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *Proc. of the 17th Int. Conf. on Automated Deduction*, pages 236–255, 2000.
- [14] T.A. Henzinger and N. Piterman. Solving games without determinization. In *Proc. 15th CSL*, LNCS 4207, pages 394–410. Springer, 2006.
- [15] G.J. Holzmann. *The Spin Model Checker: primer and reference manual*. Addison-Wesley, 2004.
- [16] B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Proc 19th CAV*, LNCS 4590, pages 258–262, 2007.
- [17] H.W. Kuhn. Solvability and consistency for linear equations and inequalities. *The American Mathematical Monthly*, 63(4):217–232, 1956.
- [18] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. EATCS Monographs on Theoretical Computer Science. Springer, 1986.
- [19] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.
- [20] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.
- [21] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
- [22] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.