Properties and Utilization of Capacitated Automata

Orna Kupferman[1] and Tami Tamir[2]

1 **School of Engineering and Computer Science, Hebrew University, Israel. E-mail: orna@cs.huji.ac.il.**
2 **School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. E-mail: tami@idc.ac.il.**

Abstract

We study *capacitated automata* (CAs), where transitions correspond to resources and may have bounded capacities. Each transition in a CA is associated with a (possibly infinite) bound on the number of times it may be traversed. We study CAs from two points of view. The first is that of traditional automata theory, where we view CAs as recognizers of formal languages and examine their expressive power, succinctness, and determinization. The second is that of resource-allocation theory, where we view CAs as a rich description of a flow network and study their utilization.

1998 ACM Subject Classification F.4.3 : Formal Languages, B.8.2 Performance Analysis and Design Aids, F.1.1 Models of Computation

Keywords and phrases Automata, Capacitataed transitions, Determinization, Maximum utilization

1 Introduction

Finite state automata are used in the modeling and design of finite-state systems and their behaviors, with applications in engineering, databases, linguistics, biology, and many more. The traditional definition of an automaton does not refer to its transitions as consumable resources. Indeed, a run of an automaton is a sequence of successive transitions, and there is no bound whatsoever on the number of times that a transition may be traversed. In practice, the use of a transition may correspond to the use of some resource. For example, it may be associated with the usage of some energy-consuming machine, application of some material, or consumption of bandwidth. We study *capacitated automata* (CAs). In this model, transitions correspond to resources and may have bounded capacities. Formally, each transition is associated with a (possibly infinite) integral bound on the number of times it may be traversed. A word w is accepted by a nondeterministic capacitated automaton (NCA) $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on w; one that reaches an accepting state and respects the bounds on the transitions.

We examine CAs from two points of view. The first, more related to traditional automata theory, views CAs as recognizers of formal languages. The interesting questions that arise in this view are similar to classical questions about automata: their expressive power, succinctness, determinization, decision problems, etc. The second view, more related to traditional resource-allocation theory, views CAs as labeled flow networks. The interesting questions then have to do with optimal utilization of the system modeled by the CA.

Let us start with the first view. In terms of expressive power, we show that capacities can be removed. Thus, NCAs are not more expressive than nondeterministic finite automata (NFAs), and can recognize exactly all regular languages. The main questions that arise,

then, refer to the succinctness of the capacitated model with respect to the standard one, as well as the blow-up involved in their determinization. Consider, for example, the language $L_{n,m}$ over the alphabet $\Sigma_n = \{1, \ldots, n\}$ that contains exactly all words in which each letter in Σ_n appears at most m times. It is not hard to see that a traditional, possibly nondeterministic, automaton for $L_{n,m}$ needs at least m^n states. On the other hand, a deterministic capacitated automaton (DCA) for L_n consists of a single state with n self-loops, each labeled by a different letter from Σ_n and has capacity m. Hence, both NCAs and DCAs may be exponentially more succinct than NFAs. The nondeterministic model, however, is much stronger, and determinization involves a blow-up that is not only exponential in the state space but also linear in the product of the capacities. Note that the latter is at least exponential in the number of transitions. This is surprising, as we do allow the obtained DCA to have capacities. As we show, there are languages, in particular strongly liveness languages [1], for which the power of capacities is significant in the nondeterministic model but is not realized in the deterministic one.

We then turn to solve decision problems about CAs. Two classical problems are the *nonemptiness* (does $\mathcal{A}$ accept at least one word?) and the *membership* (does $\mathcal{A}$ accept a given word w?) problem. In the traditional model, the problems are essentially the same: checking whether $\mathcal{A}$ accepts w can be reduced to checking the emptiness of the product of $\mathcal{A}$ with w. Accordingly, both problems can be reduced to reachability. In the capacitated model, taking the product of a word with an automaton may require tracing the history of traversals, and indeed we prove that while the nonemptiness problem is not more difficult than in the traditional model, membership becomes NP-complete for NCAs. Moreover, if we augment the nonemptiness problem to consider words from a given language (that is, given a CA $\mathcal{A}$ and a regular language L, decide whether $\mathcal{A}$ accepts some word from L), it becomes NP-complete already for DCAs, even when L is given by a DFA.

We continue to the second view, where CAs model labeled flow networks. In order to motivate this view, let us first demonstrate the different ways in which we consider CAs in the two views. Consider the containment problem for CAs, asking whether a given set S of words is contained in the language of a CA $\mathcal{A}$. We can think of two variants of the problem. In the first, which corresponds to our first view, we ask whether $\mathcal{A}$ accepts w for each word $w \in S$. In the second, which corresponds to our second view, we ask whether $\mathcal{A}$ *mutually* accepts all words in S. That is, whether $\mathcal{A}$ has enough capacity to process all the words in S mutually. It is not surprising that a CA may contain S in the first view but not in the second. Consider for example the NCA $\mathcal{A}$ described in Figure 1. Let $S = \{ab, abc, ac, abb, abcb\}$. Clearly, all the words in S are accepted by $\mathcal{A}$, thus $\mathcal{A}$ contains S in the first view. On the other hand, there is no way for $\mathcal{A}$ to mutually accept all the words in S. Indeed, it is not hard to see that $\mathcal{A}$ can make only a single use of the edge $\langle q_1, c, q_2 \rangle$ even though its capacity is 2, whereas S contains three words with the letter c. Note that $\mathcal{A}$ can mutually accept the set $S' = \{ab, ac, abb, abcb\}$, but one has to carefully resolve nondeterminism in order to do it. For example, once $\mathcal{A}$ processes abb via q_1, it can no longer process both ac and $abcb$. Thus, in the second view, the challenge is to find ways to mutually accept in a given NCA as many words as possible, as we formally define below.

A natural problem that arises when reasoning about systems that consist of resources with limited capacities is to utilize these resources in the best way. In our model, the *max-utilization problem* is defined as follows. Given a CA $\mathcal{A}$, return a multiset W of words, such that $\mathcal{A}$ mutually accepts all the words in W, and $|W|$ is maximal. The max-utilization problem can be viewed as a generalization of the max-flow problem in networks [11]. In the max-flow problem, the network is utilized by units of flow, each routed from the source to

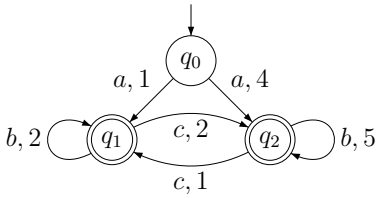

Figure 1 An NCA that contains $S = \{ab, abc, ac, abb, abcb\}$ in the traditional view but does not contain S mutually.

the target. The CA model enables a rich description of the feasible routes. The labels along a path correspond to a sequence of applications of resources. In particular, paths from an initial state to a final state correspond to feasible such sequences, and the goal is to mutually process as many of them as possible.

Sometimes, not all the sequences feasible in the NCA are desirable. Accordingly, we also consider the *max restricted-utilization problem*, where the input to the problem also includes a language specifying the desirable sequences. Then, the words in the multiset W must belong to this language. For example, the language may restrict the length of the sequences, preventing long sequences from consuming the system (see [17] for an analogous restriction in flow networks), it may restrict the number of different resources applied in a sequence, it may require an application of specific resources, it may require a specific event to trigger another specific event, and so on. Thus, while in traditional flow problems the specification of desired routes is given by means of a source and a target, the max restricted utilization problem enables specifications that are much richer than reachability. A similar lifting of reachability was studied in [3], where network formation games were extended to automata formation games. We study the complexity of the max utilization problems and show that while the unrestricted variant can be solved in polynomial time by a simple reduction to a network maximum-flow problem, even simple restrictions on the desirable routes make the problem hard to approximate. Essentially, this follows from the fact that the basic idea of an augmenting path in a network cannot be adopted to NCAs. Indeed, in NCAs every path corresponds to a sequence of applications of recourses, and the augmenting path need not correspond to a desired such sequence.

Related Work

Many extensions of automata in which transitions are augmented by numerical values have been studied. Most notable are *probabilistic automata* [19], where the values form a distribution on the successor state, and *weighted automata* [10], where weights are used in order to model costs, rewards, certainty, and many more. The semantics of these models is multi-valued: each traversal of a transition updates some accumulated value, and the language of the automaton maps words into some domain. In particular, the B and S automata of [8] count traversals on transitions. Our semantics, on the other hand, maintains the Boolean nature of regular languages, and only augments the way in which acceptance is defined.

More related to our work are extensions of automata that stay in the Boolean domain, in particular ones whose semantics involves counting. *Parikh automata* were introduced and studied by Klaedtke and Rueß in [16]. Their semantics involves counting of the number of occurrences of each letter in Σ in the word. Essentially, a Parikh automaton is a pair

$\langle \mathcal{A}, C\rangle$, where $\mathcal{A}$ is an NFA over Σ, and $C \subseteq \mathbb{N}^\Sigma$ is a set of "allowed occurrences". A word w is accepted by $\langle \mathcal{A}, C\rangle$ if both $\mathcal{A}$ accepts w and the Parakh's commutative image of w, which maps each letter in Σ to its number of occurrences in w, is in C. It is easy to see that the expressive power of Parikh automata goes beyond regular language. For example, Parikh automata can recognize the language $\{a^i b^i : i \geq 1\}$ by defining $\mathcal{A}$ to recognize a^*b^* and defining C to contain all pairs $\langle i, i\rangle$, for $i \geq 1$. In fact, by [15], Parikh automata are as expressive as reversal-bounded counter machines [13].

Several variants of Parikh automata have seen studied. In particular, [7] studied *constrained automata*, a variant that counts traversals of transitions and requires the vector of counters to belong to C, now a semilinear set of allowed vectors. NCAs can be viewed as a special case of constrained automata in which C is downwards closed. This significantly restricts the expressive power of constrained automata, and indeed the types of questions we consider are different than these studied for Parikh automata and their variants.

Additional strictly more expressive models include *multiple counters automata* [9], where transitions can be taken only if guards referring to traversals so far are satisfied, and *queue-content decision diagrams*, which are used to represent queue content of FIFO-channel systems [5, 6]. Finally, a model with the same name – *finite capacity automata* is used in [18] in order to model the control of an automated manufacturing system. This model is different from our CAs and is more related to Petri nets.

The above works take the first view on automata, namely study them as recognizers of languages. As for the second view, a lot of research has been done on optimal utilization of limited resources. In particular, as discussed above, max-utilization of a CA corresponds to a maximum flow in a network [11]. By restricting the domain from which accepted words can be chosen, we get an intractable problem. This resemble the intractability of some variants of the max-flow problem, such as k-bounded flow [4], or max-flow with bounded-length paths [17].

2 Preliminaries

A *nondeterministic finite automaton* (NFA, for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, F\rangle$, where Σ is a finite alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, and $F \subseteq Q$ is a set of final states. Given a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_l$, a *run* of $\mathcal{A}$ on w is a sequence r of successive transitions in Δ that reads w and starts in a transition from the set of initial states. Thus, $r = \langle q_0, \sigma_1, q_1\rangle, \langle q_1, \sigma_2, q_2\rangle, \ldots, \langle q_{l-1}, \sigma_l, q_l\rangle$, for $q_0 \in Q_0$. The run is accepting if $q_l \in F$. We sometimes refer to the transitions function $\delta : Q \times \Sigma \to Q$ induced by Δ, thus $q' \in \delta(q, \sigma)$ iff $\Delta(q, \sigma, q')$. The NFA $\mathcal{A}$ *accepts* the word w iff it has an accepting run on it. Otherwise, $\mathcal{A}$ *rejects* w. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$ is the set of words that $\mathcal{A}$ accepts. If $|Q_0| = 1$ and for all $q \in Q$ and $\sigma \in \Sigma$ there is at most one $q' \in Q$ with $\Delta(q, \sigma, q')$, then $\mathcal{A}$ is *deterministic*. Note that a deterministic finite automaton (DFA) has at most one run on each word.

A *nondeterministic capacitated automaton* (NCA, for short) is an NFA in which each transition has a *capacity*, bounding the number of times it may be traversed. A transition may not be bounded, in which case its capacity is ∞. Let $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$ and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. Formally, an NCA is a pair $\langle \mathcal{A}, c\rangle$, where $\mathcal{A}$ is an NFA and $c : \Delta \to \mathbb{N}^\infty$ is a capacity function that maps each transition in Δ to its capacity. A run of $\langle \mathcal{A}, c\rangle$ is a run of $\mathcal{A}$ in which the number of occurrences of each transition $e \in \Delta$ is at most $c(e)$. When $\mathcal{A}$ is deterministic, then so is $\langle \mathcal{A}, c\rangle$. For a CA $\mathcal{A}$ and a set of words S, we say that $\mathcal{A}$ *mutually accepts* S if there are $|S|$ accepting runs, one for each word in S, such that for each transition $e \in \Delta$,

the total number of occurrences of e in all these runs is at most $c(e)$.

For a capacity function $c : \Delta \to \mathbb{N}^\infty$, let $c_\downarrow$ be the set of capacity functions obtained by closing c downwards. Formally, a function $c' : \Delta \to \mathbb{N}^\infty$ is in $c_\downarrow$ if for all transitions $e \in \Delta$ with $c(e) = \infty$, we have $c'(e) = \infty$, and for all transitions $e \in \Delta$ with $c(e) \in \mathbb{N}$, we have $0 \leq c'(e) \leq c(e)$. It is easy to see that the size of $c_\downarrow$, denoted $|c_\downarrow|$, is $\Pi_{e:c(e)\in\mathbb{N}^+}(c(e)+1)$. Thus, $|c_\downarrow|$ is exponential in the number of transitions with bounded capacities.

3 Theoretical Properties of NCAs

In this section we study the expressive power and succinctness of NCAs with respect to NFAs, as well as their determinization.

3.1 Capacities Removal

▶ **Theorem 1.** *NCAs accept regular languages: Every NCA $\langle \mathcal{A}, c\rangle$ has an equivalent NFA $\mathcal{A}'$. The size of $\mathcal{A}'$ is linear in the size of $\mathcal{A}$ and $|c_\downarrow|$.*

Proof. Given an NCA $\langle \mathcal{A}, c\rangle$ with $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, F\rangle$, we define an equivalent NFA $\mathcal{A}' = \langle \Sigma, Q', Q'_0, \Delta', F'\rangle$ as follows.

- $Q' = Q \times c_\downarrow$ and $Q'_0 = Q_0 \times \{c_\downarrow\}$. That is, each state in $\mathcal{A}'$ maintains both the corresponding state in $\mathcal{A}$ and the capacities that are left to be consumed.
- For $q, q' \in Q$, $d, d' \in c_\downarrow$, and $\sigma \in \Sigma$, we have that $\Delta'(\langle q, d\rangle, \sigma, \langle q', d'\rangle)$ iff $\Delta(q, \sigma, q')$, $d(\langle q, \sigma, q'\rangle) > 0$, $d'(e) = d(e)$ for all $e \neq \langle q, \sigma, q'\rangle$, and $d'(\langle q, \sigma, q'\rangle) = d(\langle q, \sigma, q'\rangle) - 1$. That is, d' is updated to take the traversal of $\langle q, \sigma, q'\rangle$ into an account by reducing its capacity by 1.
- $F' = F \times c_\downarrow$.

Note that the construction preserves determinism, thus if $\mathcal{A}$ is a DCA, the obtained $\mathcal{A}'$ is a DFA. It is not hard to prove that each run r of $\mathcal{A}$ corresponds to a run of $\mathcal{A}'$, obtained by pairing each state in r by the capacity function that reflects the updates to c according to the transitions traversed so far. Dually, each run r' of $\mathcal{A}'$ corresponds to a run of $\mathcal{A}$, obtained by projecting r' on the Q-elements of its states. By the definition of $\mathcal{A}'$, the obtained run respects the bounds on the transitions.

We prove that the blow-up in $|c_\downarrow|$ cannot be avoided. We prove it already for single-state DCAs. Given two parameters $n, m \in \mathbb{N}$, consider the DCA $\langle \mathcal{A}_{n,m}, c_{n,m}\rangle$, where $\mathcal{A}_{n,m} = \langle \{1, \ldots, n\}, \{q\}, \{q\}, \Delta, \{q\}\rangle$ is such that each letter $i \in \{1, \ldots, n\}$ contributes to Δ the transition $\langle q, i, q\rangle$. That is, $\mathcal{A}_{n,m}$ consists of a single state with one self-loop transition for each of the n letters. The capacity of all transitions is m. Thus, $c_{n,m}(\langle q, i, q\rangle) = m$ for all $i \in \{1, \ldots, n\}$. It is easy to see that the language $L_{n,m}$ of $\langle \mathcal{A}_{n,m}, c_{n,m}\rangle$ is the set of all words in which each of the letters $\{1, \ldots, n\}$ appears at most m times, and that every NFA that recognizes $L_{n,m}$ needs at least n^m states. ◀

Theorem 1 implies that, like regular languages, NCAs and DCAs are closed under union, intersection, and complementation, and that NCAs can be determinized. The question is the blow-up involved in the corresponding constructions, in particular whether they need to involve removal of capacities.

3.2 Determinization

In this section we study the succinctness of NCAs with respect to DCAs. We show that NCAs are exponentially more succinct not only in the number of states but also in the number of transitions. More precisely, determination may involve a blow-up linear in $c_{\downarrow}$. This is surprising, as we do allow the obtained deterministic automaton to be capacitated. We first prove that there are languages for which capacities are not useful in the deterministic setting. A language $L \subseteq \Sigma^*$ is *strongly liveness* if $L = \Sigma^* \cdot L$ [1]. Thus, in terms of temporal logic, strongly liveness languages correspond to properties if the form "eventually ψ" for some behavior ψ.

▶ **Lemma 2.** *A DCA for a strongly liveness language L is not smaller than a DFA for L.*

Proof. Consider a DCA $\langle \mathcal{A}, c \rangle$ that recognizes L. We say that a word $w \in \Sigma^*$ *consumes* $\langle \mathcal{A}, c \rangle$ if, after reading w, a run of $\mathcal{A}$ can proceed only along transitions with infinite capacity (or cannot proceed at all). It is easy to see that there exists at least one word w that consumes $\langle \mathcal{A}, c \rangle$. Let q be the state that $\mathcal{A}$ reaches by reading w. Since $L = \Sigma^* \cdot L$, we have that $w \cdot L \subseteq L$. Hence, since $\mathcal{A}$ is deterministic, the language of the DFA $\mathcal{A}'$ obtained from $\mathcal{A}$ by making q its initial state and by removing all transitions that do not have infinite capacity is L. The size of $\mathcal{A}'$ is at most the size of $\mathcal{A}$, and we are done. ◀

▶ **Theorem 3.** *Determinization of NCAs involves a blow-up exponential in Q and linear in $c_{\downarrow}$.*

Proof. The exponential blow-up with respect to Q follows from the known exponential blow-up in determinization of NFAs [20]. In order to prove the blow-up in $c_{\downarrow}$, we describe a family $L'_{n,m}$, for $n, m \geq 1$ of strongly liveness languages such that $L'_{n,m}$ is over the alphabet $\Sigma_n = \{0, \ldots, n\}$, it can be recognized by a two-state NCA with n transitions with capacity m. By Lemma 2, the size of a DCA for $L'_{n,m}$ is equal to the size of a DFA for it, which is at least m^n.

Let $L_{n,m}$ be the language of all words w over $\{1, \ldots, n\}$ such that each letter in $\{1, \ldots, n\}$ appears in w at most m times (the same language as in the proof of Theorem 1). We define $L'_{n,m} = \Sigma_n^* \cdot 0 \cdot L_{n,m}$, Clearly, $L'_{n,m}$ is a strongly liveness language, and it can be recognized by the two-state NCA described in Figure 2.

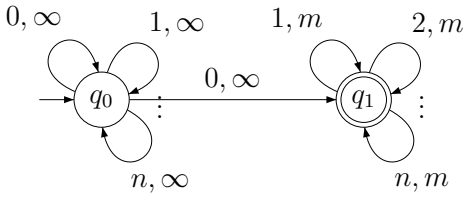

Figure 2 An NCA for $L'_{n,m}$.

◀

4 Decision Problems

In this section we study the following decision problems for NCAs and DCAs. Below we also state their known complexity in the traditional setting (see, for example [12]).

- The *nonemptiness* problem: given an automaton $\mathcal{A}$, decides whether $L(\mathcal{A}) \neq \emptyset$. For both NFA and DFA, the nonemptiness problem is NLOGSPACE-complete.

- The *membership* problem: given an automaton $\mathcal{A}$ and a finite word w, decide whether $w \in L(\mathcal{A})$. For NFA and DFA, the membership problem is NLOGSPACE-complete and LOGSPACE-complete, respectively.
- The *relative nonemptiness* problem: given an automaton $\mathcal{A}$ and a language L, decide whether $\mathcal{A}$ accepts at least one word from L. The relative nonemptiness problem for an NFA or a DFA $\mathcal{A}$ and a language L given by an NFA or a DFA is NLOGSPACE-complete.

▶ **Theorem 4.** *The nonemptiness problem for NCAs and DCAs is NLOGSPACE-complete.*

Proof. An NCA is nonempty iff there is a simple path from some initial state to some accepting state. Since the path is simple, capacities do not play a role (beyond exclusion of transitions with capacity 0). Thus, nonemptiness can be reduced to reachability, implying membership in NLOGSPACE. The lower bound follows from NLOGSPACE hardness for DFA emptiness. ◀

▶ **Theorem 5.** *The membership problem can be solved in linear time for DCAs and is NP-complete for NCAs.*

Proof. We start with the upper bounds. Given a DCA $\langle \mathcal{A}, c\rangle$ and a word w, we can trace the single run of $\langle \mathcal{A}, c\rangle$ on w and check that it ends in an accepting state and respects c. When $\langle \mathcal{A}, c\rangle$ is an NCA, a witness to the membership of w in $\langle \mathcal{A}, c\rangle$ is an accepting run of $\langle \mathcal{A}, c\rangle$ on w. As above, it can be checked in linear time.

For NCAs, we prove hardness in NP already for the case $|\Sigma| = 1$. We describe a reduction from the problem of deciding whether a given directed graph has a Hamiltonian cycle – one that visits all vertices of the graph exactly once. Given a graph $G = \langle V, E\rangle$, we construct the NCA $\langle \mathcal{A}, c\rangle$ as follows (see an example in Figure 3. The graph G is on the left, the NCA $\mathcal{A}$ is in the middle). Let v_1 be some vertex in V. Then, $\mathcal{A} = \langle \{\sigma\}, V \times \{\text{in,out}\}, \{\langle v_1, \text{out}\rangle\}, \Delta, \{\langle v_1, \text{out}\rangle\}\rangle$ is such that each vertex $v \in V$ contributes to Δ the transition $\langle \langle v, \text{in}\rangle, \sigma, \langle v, \text{out}\rangle\rangle$ with capacity 1, and each edge $\langle u, v\rangle$ in E contributes to Δ the transition $\langle \langle u, \text{out}\rangle, \sigma, \langle v, \text{in}\rangle\rangle$, again with capacity 1. It is not hard to see that G has a Hamiltonian cycle iff $\langle \mathcal{A}, c\rangle$ accepts the word $\sigma^{2|V|}$. Indeed, having capacity 1 on the transitions that correspond to vertices in V guarantees that each vertex is visited at most once, thus a word of length $2|V|$ must close a cycle and visits exactly all vertices in G. ◀

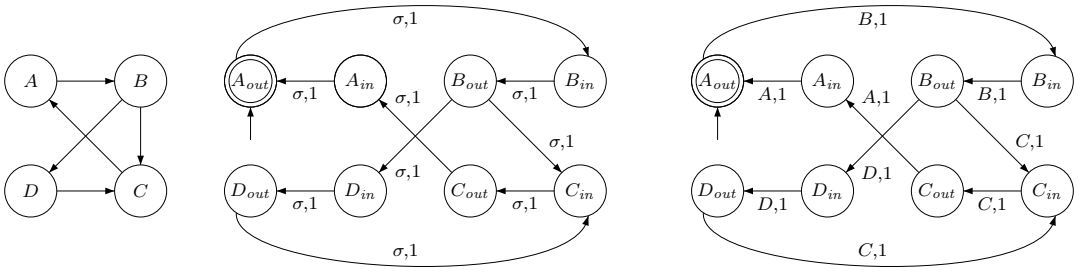

Figure 3 Two reductions from the Hamiltonian cycle problem.

▶ **Theorem 6.** *The relative nonemptiness problem for NCAs and DCAs relative to languages given by NCAs, DCAs, NFAs, or DFAs is NP-complete.*

Proof. Consider a capacitated automaton $\langle \mathcal{A}, c\rangle$ and an automaton $\mathcal{U}$ such that we want to check the nonemptiness of $\mathcal{A}$ with respect to $L(\mathcal{U})$. Note that we have eight cases to consider, reflecting whether $\mathcal{A}$ is an NCA or a DCA and whether $\mathcal{U}$ is an NCA, DCA, NFA, or DFA. We prove that all eight cases are NP-complete.

We start with membership in NP for the most general case, where both $\langle \mathcal{A}, c\rangle$ and $\mathcal{U} = \langle \mathcal{A}', c'\rangle$ are NCAs. A witness to the relative nonemptiness is a word w and accepting runs of $\langle \mathcal{A}, c\rangle$ and $\langle \mathcal{A}', c'\rangle$ on it. The word w does not traverse cycles in the product of $\mathcal{A}$ and $\mathcal{A}'$. It is thus of polynomial length in the product, which is of size $|\mathcal{A}| \cdot |\mathcal{A}'|$. Checking that the runs respect c and c' can also be done in polynomial time.

We prove hardness in NP for the most restricted case, where $\langle \mathcal{A}, c\rangle$ is a DCA and $\mathcal{U}$ is a DFA. We describe a reduction from the problem of deciding whether a given directed graph has a Hamiltonian cycle (see an example in Figure 3. The graph G is on the left, the NCA $\mathcal{A}$ is on the right). Given $G = \langle V, E\rangle$ with $V = \{v_1, \ldots, v_n\}$, we construct $\mathcal{A} = \langle V, V \times \{\text{in},\text{out}\}, \{\langle v_1, \text{out}\rangle\}, \Delta, \{\langle v_1, \text{out}\rangle\}\rangle$, where v_1 is an arbitrary vertex in V. Each vertex $v \in V$ contributes to Δ the transition $\langle\langle v, \text{in}\rangle, v, \langle v, \text{out}\rangle\rangle$ with capacity 1, and each edge $\langle u, v\rangle$ in E contributes to Δ the transition $\langle\langle u, \text{out}\rangle, v, \langle v, \text{in}\rangle\rangle$ with capacity 1. It is easy to see that $\mathcal{A}$ is indeed a DCA and that G has a Hamiltonian cycle iff $\mathcal{A}$ accepts a word in the language $[(v_1 \cdot v_1) + \cdots + (v_n \cdot v_n)]^n$, which can be recognized by a DFA with $O(n^2)$ states. In the example, the Hamiltonian cycle $ABDCA$ corresponds to the word $BBDDCCAA$. ◀

5 The Maximum Utilization Problem

A natural problem that arises when reasoning about systems that consist of resources with limited capacities is to utilize these resources in the best way. In our model, the maximum utilization problem is defined as follows. Given a CA $\langle \mathcal{A}, c\rangle$, return a multiset W of words, such that $\langle \mathcal{A}, c\rangle$ has enough capacity to mutually accept all the words in W, and $|W|$ is maximal. We refer to W as an *optimal utilization mutiset* for $\langle \mathcal{A}, c\rangle$.

As discussed in Section 1, the max-utilization problem can be viewed as a generalization of the max-flow problem in a network. The CA model enables a rich description of the feasible routes. Sometimes, not all the sequences allowed by the CA are desirable. Accordingly, we also consider the *max restricted utilization problem*, where the input to the problem also includes a language L, specifying the desirable sequences. Then, the words in the optimal utilization multiset must belong to L. In Section 5.1, we show that the unrestricted max-utilization problem can be solved in polynomial time by a reduction to the classical network-flow problem [11]. In section 5.2 we study the restricted case and show that adding restrictions makes the problem much more complex.

5.1 Maximum Unrestricted Utilization

We present an optimal algorithm for the max-utilization problem. The algorithm is based on a reduction to a max-flow problem in a network. Recall that a max-flow problem is defined over a flow-network given by a directed graph $G = \langle V, E\rangle$, two vertices $s, t \in V$ designated as the source and the target vertices, and a capacity function c that maps each edge $e \in E$ to a positive integral capacity $c(e)$. For a vertex v, let $in(v)$ and $out(v)$ denote the set of edges into and out of v, respectively. A *legal flow* is a function $f : E \to \mathbb{R}$ such that for every edge $e \in E$, it holds that $0 \leq f(e) \leq c(e)$, and for every vertex $v \in V \setminus \{s, t\}$, it holds that $\sum_{e \in in(v)} f(e) = \sum_{e \in out(v)} f(e)$. A max-flow is a legal flow that maximizes the flow leaving the source, given by $\sum_{e \in out(s)} f(e) - \sum_{e \in in(s)} f(e)$.

▶ **Theorem 7.** *The max-utilization problem for NCAs and DCAs can be solved in polynomial time.*

Proof. Let $\langle \mathcal{A}, c\rangle$ be an NCA, with $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, F\rangle$. If $\mathcal{A}$ has a path from Q_0 to F all whose transitions have infinite capacity (in particular, if $Q_0 \cap F \neq \emptyset$), then, as the optimal utilization multiset is not restricted, so is the max-flow, and we return as a witness the (possibly empty) word along the above accepting run.

Otherwise, the optimal utilization multiset must be finite and we proceed as follows. Given $\langle \mathcal{A}, c\rangle$, we construct a flow-network $G = \langle Q \cup \{s, t\}, E\rangle$, where every transition $\langle p, \sigma, q\rangle \in \Delta$ induces an edge $\langle p, q\rangle$ in E with capacity $c(\langle p, \sigma, q\rangle)$. The set E includes also the set of edges $\{s\} \times Q_0$ and $F \times \{t\}$, all with an unbounded capacity. It is easy to see that a max-flow in G corresponds to a maximal multiset of words that can be mutually processed by $\langle \mathcal{A}, c\rangle$. The optimal utilization multiset W can be obtained by tracking the (s, t)-paths in the max-flow. Note that since there are no restriction on the words in W, the alphabet in $\langle \mathcal{A}, c\rangle$ plays no role. ◀

5.2 Maximum Restricted Utilization

In the max restricted-utilization problem, we are given, in addition to a CA $\langle \mathcal{A}, c\rangle$, also a regular language L. The optimal utilization multi set W then has to contain only words from L. We refer to L as the *restricting language*.

We present a polynomial-time optimal algorithm for the max restricted-utilization problem for the case the restricting language consists of words of length at most 2. This is indeed a very limited class of languages. Yet, it is tight, as we provide an APX-hardness proof already for the case that $\langle \mathcal{A}, c\rangle$ is a DCA with $c \equiv 1$ and the restricting language consists of words of length 3.

▶ **Theorem 8.** *When all the words in the restricting language are of length at most* 2*, the max restricted-utilization problem can be solved in polynomial time.*

Proof. Let $\langle \mathcal{A}, c\rangle$ be an NCA and L a restricting language all of whose words are of length at most 2. First, if $\varepsilon \in L(\langle \mathcal{A}, c\rangle) \cap L$, then the optimal utilization multiset is infinite, and we are done. Hence, we assume that all words in L are of length 1 or 2. We present an optimal algorithm that is based on reducing the problem to a *maximum b-matching* problem. An instance of b-matching consists of an undirected graph $G = \langle V, E\rangle$ and a budget function $b : V \to \mathbb{N}$. A b-matching is an assignment of a non-negative integer weight x_e to every edge $e \in E$, such that for every vertex $v \in V$, the sum of weights on edges incident with v is at most $b(v)$. The maximum b-matching problem is to find a matching of maximum profit; that is, $\max \sum_e x_e$. The graph G need not be simple. That is, self-loops and parallel edges are allowed. By [2], the maximum b-matching problem can be solved in time polynomial in $|V|$ and $|E|$.

Let $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, F\rangle$, and let $W_1(L)$ and $W_2(L)$ denote the words of length 1 and 2 in L, respectively. For every $\sigma \in W_1(L)$, let $A_\sigma \subseteq \Delta$ be the set of transitions that form an accepting run on σ in $\langle \mathcal{A}, c\rangle$. Formally, $e \in A_\sigma$ iff $e = \langle q_0, \sigma, p\rangle$ for $q_0 \in Q_0$, $p \in F$, and $c(e) \geq 1$. For every $w = \sigma_1 \cdot \sigma_2 \in W_2(L)$, let $A_w \subseteq \Delta \times \Delta$ be the set of pairs of transitions that form an accepting run on w in $\mathcal{A}$. Formally, $\langle e_1, e_2\rangle \in A_w$ iff $e_1 = \langle q_1, \sigma_1, p_1\rangle, e_2 = \langle q_2, \sigma_2, p_2\rangle$ for $q_1 \in Q_0$, $p_1 = q_2$, $p_2 \in F$, $c(e_1) \geq 1$, and $c(e_2) \geq 1$. If $e_1 = e_2$, then we also require $c(e_1) \geq 2$. Note that if $w \notin L(\mathcal{A})$, then E_w is empty. Also, if w has several accepting runs in $\mathcal{A}$, then $|E_w| > 1$. It is possible to compute the above sets in time $O(|Q|^2)$.

We construct an undirected graph $G = \langle \Delta \cup \{v_0\}, E\rangle$, where E consists of edges of two types, defined as follows.

1. $\langle e_i, v_0\rangle \in E$ iff there exists a word $\sigma \in W_1(L)$ such that $e_i \in A_\sigma$.
2. $\langle e_i, e_j\rangle \in E$ iff there exists a word $w \in W_2(L)$ such that $\langle e_i, e_j\rangle \in A_w$.

Every edge of type (e_i, v_0) corresponds to a 1-letter word. Every edge of type (e_i, e_j) corresponds to at least one 2-letter word. Since Q_0 may include several initial states, it is possible that both $\langle e_i, e_j\rangle$ and $\langle e_j, e_i\rangle$ form accepting runs in $\langle \mathcal{A}, c\rangle$. It is also possible to have a self-loop in G if $\mathcal{A}$ includes a self-loop from the initial state.

An example of the reduction is given in Figure 4. The CA $\langle \mathcal{A}, c\rangle$ is on the left and the b-matching instance constructed for $L = \{a, aa, ab, ac, ba, bc, cc\}$ is on the right. Note that each of the words a, aa, and ab has two possible accepting runs in $\langle \mathcal{A}, c\rangle$. Thus, each of these words induces two edges in G. Words that belong to $L \setminus L(\langle \mathcal{A}, c\rangle)$, like ba and cc, as well as words that belong to $L(\langle \mathcal{A}, c\rangle) \setminus L$, like b or bb, do not induce an edge in G.

To complete the definition of the b-matching instance, we set the vertex budgets as follows. For every $e_i \in \Delta$, we set $b(e_i) = c_i$. For v_0, we set $b(v_0) = \infty$.

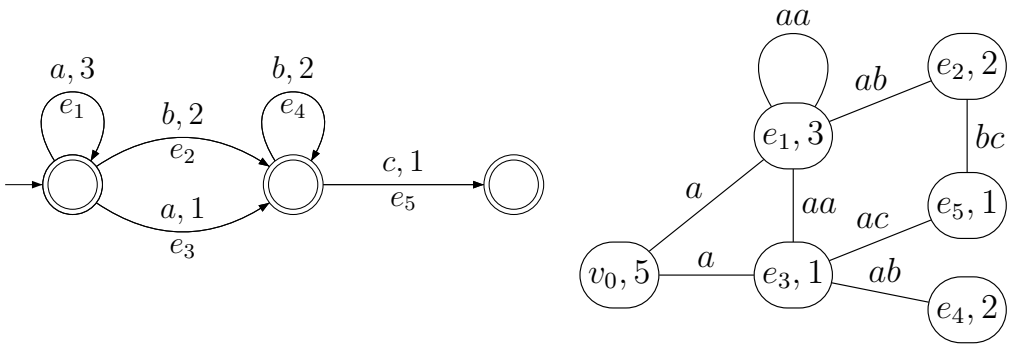

Figure 4 An NCA and the b-matching instance constructed for $L = \{a, aa, ab, ac, ba, bc, cc\}$.

Consider a feasible b-matching in G. By the construction of G, every edge in E corresponds to a word in $L(\langle \mathcal{A}, c\rangle) \cap L$. The budget constraints on the vertices correspond to the edge-capacities. Thus, every set of words from L that can be mutually accepted by $\langle \mathcal{A}, c\rangle$ corresponds to a feasible b-matching in G and vice-versa. In particular, a maximum b-matching corresponds to an optimal utilization multiset that is contained in L. ◀

Next, we show that the above is the most positive result we can achieve for the max restricted-utilization problem. That is, when L may include three-letter words, the max-utilization problem becomes APX-hard. That is, there exists a constant c such that it is NP-hard to find an approximation algorithm with approximation ratio better than c.

▶ **Theorem 9.** *The max restricted-utilization problem is APX-hard. This is valid already when the restricting language consists only of words of length* 3 *and a unit capacity DCA.*

Proof. We show an approximation-preserving reduction from the maximum 3-bounded 3-dimensional matching problem (3DM-3). The input to the 3DM-3 problem is a set of triplets $T \subseteq X \times Y \times Z$, where $|X| = |Y| = |Z| = n$. The number of occurrences of every element of $X \cup Y \cup Z$ in T is at most 3. The number of triplets is $|T| \geq n$. The desired output is a 3-dimensional matching in T of maximal cardinality; i.e., a subset $T' \subseteq T$, such that every element in $X \cup Y \cup Z$ appears at most once in T', and $|T'|$ is maximal. Kann showed in [14] that 3DM-3 is APX-hard.

Given an instance of 3DM-3, we construct a DCA $\langle \mathcal{A}, c\rangle$, with $\mathcal{A} = \langle \Sigma, Q, Q_0, \Delta, F\rangle$, as follows. First, $\Sigma = X \cup Y \cup Z, Q = \{q_0, q_1, q_2, q_3\}, Q_0 = \{q_0\}, F = \{q_3\}$. The transition

relation Δ consists of $3n$ transitions all having capacity 1. There are n parallel transitions $\langle q_0, x_i, q_1\rangle$ for all $1 \leq i \leq n$, n parallel transitions $\langle q_1, y_j, q_2\rangle$ for all $1 \leq j \leq n$, and n parallel transitions $\langle q_2, z_k, q_3\rangle$ for all $1 \leq k \leq n$. The capacity of all transitions is 1.

To complete the reduction, we define the restricting language $L = \{x_i{\cdot}y_j{\cdot}z_k : \langle x_i, y_j, z_k\rangle \in T\}$. Note that the reduction is polynomial. Also, since the 3DM instance is 3-bounded, we have that $|L| = O(n)$.

Let $W \subseteq L$ be a set of words that can be mutually accepted by $\mathcal{A}$. The unit capacities imply that every element in $X \cup Y \cup Z$ appears in at most one word in W. Thus, every matching in T corresponds to a possible utilization multiset W. In particular, a maximum matching corresponds to an optimal utilization multiset for $\langle \mathcal{A}, c\rangle$, contained in L. ◀

▶ Remark. In the *weighted max (possibly restricted) utilization* problem, there is a profit function $p : \Sigma^* \to \mathbb{R}$ that associates profit with each word. The profit function can be given, for example, by a weighted automaton. The optimal utilization set is now a set $W \subseteq \Sigma^*$ that can be mutually accepted by $\langle \mathcal{A}, c\rangle$ and for which $\sum_{w\in W} p(w)$ is maximal. In the restricted variant, we require $W \subseteq L$.

Clearly, the lower bounds we prove apply also to the weighted version. As we now show, the polynomial upper bound for the case of a restricting language that consists only of words of length at most 2 can be extended to the weighted setting. For that, we also need a weighted version of the b-matching problem. There, every edge $e \in E$ is associated with a profit $p(e)$, and the maximum b-matching problem is to find a matching that maximizes $\sum_e p(e)x_e$. The algorithm in [2] applies also to the weighted variant.

Now, in the algorithm described in the proof of Theorem 8, we define the profits as follows. For every edge $e \in E$, we set $p(e)$ to be the profit $p(w)$ of the corresponding word. If e corresponds to two words, w_1, w_2, then set $p(w) = \max\{p(w_1, p(w_2)\}$. ◀

References

1. B. Alpern and F.B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.
2. R.P. Anstee. A polynomial algorithm for b-matching: An alternative approach. *Information Processing Letters*, 24:153–157, 1987.
3. G. Avni, O. Kupferman, and T. Tamir. Network-formation games with regular objectives. In *Proc. 17th Int. Conf. on Foundations of Software Science and Computation Structures*, LNCS 8412, pages 119–133. Springer, 2014.
4. G. Baier, E. Köhler, and M. Skutella. The k-splittable flow problem. *Algorithmica*, 42(3-4):231–248, 2005.
5. B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds. In *Proc. 8th Int. Conf. on Computer Aided Verification*, LNCS 1102, pages 1–12. Springer, 1996.
6. A. Bouajjani, P. Habermehl, and T. Vojnar. Verification of parametric concurrent systems with prioritised FIFO resource management. *Formal Methods in System Design*, 32(2):129–172, 2008.
7. M. Cadilhac, A. Finkel, and P. McKenzie. On the expressiveness of Parikh automata and related models. In *3rd Workshop on Non-Classical Models for Automata and Applications*, pages 103–119, 2011.
8. T. Colcombet, D. Kuperberg, and S. Lombardy. Regular temporal cost functions. In *Proc. 37th Int. Colloq. on Automata, Languages, and Programming*, LNCS 6199, pages 563–574, 2010.

9 H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *Proc. 10th Int. Conf. on Computer Aided Verification*, LNCS 1427, pages 268–279. Springer, 1998.

10 M. Droste, W. Kuich, and H. Vogler (eds.). *Handbook of Weighted Automata*. Springer, 2009.

11 L.R. Ford and D.R. Fulkerson. *Flows in networks*. Princeton Univ. Press, Princeton, 1962.

12 J.E. Hopcroft and R. Motwani and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison-Wesley, 2000.

13 O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.

14 V. Kann. Maximum bounded 3-dimensional matching is max-snp-complete. *Information Processing Letters*, 37(1):27–35, 1991.

15 F. Klaedtke and H. Rueß. Parikh automata and monadic second-order logics with linear cardinality constraints. Technical Report 177, Universit´at Freiburg, 2002.

16 F. Klaedtke and H. Rueß. Monadic second-order logics with cardinalities. In *Proc. 30th Int. Colloq. on Automata, Languages, and Programming*, LNCS 2719, pages 681–696. Springer, 2003.

17 A.R. Mahjoub and S.T. McCormick. Max flow and min cut with bounded-length paths: complexity, algorithms, and approximation. *Mathematical Programming*, 124(1-2):271–284, 2010.

18 R.G. Qiu and S.B. Joshi. Deterministic finite capacity automata: a solution to reduce the complexity of modeling and control of automated manufacturing systems. In *Proc. Symp. on Computer-Aided Control System Design*, pages 218 –223, 1996.

19 M. O. Rabin. Probabilistic automata. *Information and Control*, 6:230–245, 1963.

20 M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.