Automata vs Linear-Programming Discounted-Sum Inclusion: 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17,...

Chapter · July 2018

DOI: 10.1007/978-3-319-96142-2_9

CITATIONS
0

READS
10

3 authors, including:

Suguman Bansal
Rice University
5 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Moshe Vardi
Rice University
755 PUBLICATIONS **29,820** CITATIONS
SEE PROFILE

Automata vs Linear-Programming Discounted-Sum Inclusion

Suguman Bansal, Swarat Chaudhuri, and Moshe Y. Vardi

Rice University, Houston TX, TX 77005, USA

Abstract. The problem of *quantitative inclusion* formalizes the goal of comparing quantitative dimensions between systems such as worst-case execution time, resource consumption, and the like. Such systems are typically represented by formalisms such as weighted logics or weighted automata. Despite its significance in analyzing the quality of computing systems, the study of quantitative inclusion has mostly been conducted from a theoretical standpoint. In this work, we conduct the first empirical study of quantitative inclusion for discounted-sum weighted automata (DS-inclusion, in short).

Currently, two contrasting approaches for DS-inclusion exist: the linear-programming based DetLP and the purely automata-theoretic BCV. Theoretical complexity of DetLP is exponential in time and space while of BCV is PSPACE-complete. All practical implementations of BCV, however, are also exponential in time and space. Hence, it is not clear which of the two algorithms renders a superior implementation.

In this work we present the first implementations of these algorithms, and perform extensive experimentation to compare between the two approaches. Our empirical analysis shows how the two approaches complement each other. This is a nuanced picture that is much richer than the one obtained from the theoretical study alone.

1 Introduction

The analysis of quantitative dimensions of systems, such as worst-case execution time, energy consumption, and the like, has been studied thoroughly in recent times. By and large, these investigations have tended to be purely theoretical. While some efforts in this space [12,13] do deliver prototype tools, the area lacks a thorough empirical understanding of the relative performance of different but related algorithmic solutions. In this paper, we further such an empirical understanding for *quantitative inclusion* for *discounted-sum weighted automata*.

Weighted automata [17] are a popular choice for system models in quantitative analysis. The problem of quantitative language inclusion [15] formalizes the goal of determining which of any two given systems is more efficient under such a system model. In a discounted-sum weighted automata the value of quantitative dimensions are computed by *aggregating* the costs incurred during each step of a system execution with discounted-sum aggregation. The discounted-sum (DS) function relies on the intuition that costs incurred in the near future are

more "expensive" than costs incurred later on. Naturally, it is the choice for aggregation for applications in economics and game-theory [20], Markov Decision Processes with discounted rewards [16], quantitative safety [13], and more.

The hardness of quantitative inclusion for nondeterministic DS-automata, or DS-inclusion, is evident from PSPACE-hardness of language-inclusion (LI) problem for nondeterministic Büchi automata [23]. Decision procedures for DS-inclusion were first investigated in [15], and subsequently through target discounted-sum [11], DS-determinization [10]. A comparator-based argument [9] finally established its PSPACE-completeness. However, these theoretical advances in DS-inclusion have not been accompanied with the development of efficient and scalable tools and algorithms. This is the focus of this paper; our goal is to develop practical algorithms and tools for DS-inclusion.

Theoretical advances have lead to two algorithmic approaches for DS-inclusion. The first approach, referred to as DetLP, combines automata-theoretic reasoning with linear-programming (LP). This method first determinizes the DS-automata [10], and reduces the problem of DS-inclusion for deterministic DS-automata to LP [7,8]. Since determinization of DS-automata causes an exponential blow-up, DetLP yields an exponential time algorithm. An essential feature of this approach is the separation of automata-theoretic reasoning– determinization– and numerical reasoning, performed by an LP-solver. Because of this separation, it does not seem easy to apply on-the-fly techniques to this approach and perform it using polynomial space, so this approach uses exponential time and space.

In contrast, the second algorithm for DS-inclusion, referred to as BCV (after name of authors) is purely automata-theoretic [9]. The component of numerical reasoning between costs of executions is handled by a special Büchi automaton, called the *comparator*, that enables an on-line comparison of the discounted-sum of a pair of weight-sequences. Aided by the comparator, BCV reduces DS-inclusion to language-equivalence between Büchi automata. Since language-equivalence is in PSPACE, BCV is a polynomial-space algorithm.

While the complexity-theoretic argument may seem to suggest a clear advantage for the pure automata-theoretic approach of BCV, the perspective from an implementation point of view is more nuanced. BCV relies on LI-solvers as its key algorithmic component. The polynomial-space approach for LI relies on Savitch's Theorem, which proves the equivalence between deterministic and nondeterministic space complexity [21]. This theorem, however, does not yield a practical algorithm. Existing efficient LI-solvers [3,4] are based on Ramsey-based inclusion testing [6] or rank-based approaches [18]. These tools actually use exponential time and space. In fact, the exponential blow-up of Ramsey-based approach seems to be worse than that of DS-determinization. Thus, the theoretical advantage BCV seems to evaporate upon close examination. Thus, it is far from clear which algorithmic approach is superior. To resolve this issue, we provide in this paper the first implementations for both algorithms and perform exhaustive empirical analysis to compare their performance.

Our first tool, also called DetLP, implements its namesake algorithm as it is. We rely on existing LP-solver GLPSOL to perform numerical reasoning. Our sec-

ond tool, called QuIP, starts from BCV, but improves on it. The key improvement arises from the construction of an improved comparator with fewer states. We revisit the reduction to language inclusion in [9] accordingly. The new reduction reduces the transition-density of the inputs to the LI-solver (Transition density is the ratio of transitions to states), improving the overall performance of QuIP since LI-solvers are known to scale better at lower transition-density inputs [19]

Our empirical analysis reveals that theoretical complexity does not provide a full picture. Despite its poorer complexity, QuIP scales significantly better than DetLP, although DetLP solves more benchmarks. Based on these observations, we propose a method for DS-inclusion that leverages the complementary strengths of these tools to offer a scalable tool for DS-inclusion. Our evaluation also highlights the limitations of both approaches, and opens directions for further research in improving tools for DS-inclusion.

2 Preliminaries

Büchi automata A *Büchi automaton* [23] is a tuple $\mathcal{A} = (S, \Sigma, \delta, Init, \mathcal{F})$, where S is a finite set of *states*, Σ is a finite *input alphabet*, $\delta \subseteq (S \times \Sigma \times S)$ is the *transition relation*, $Init \subseteq S$ is the set of *initial states*, and $\mathcal{F} \subseteq S$ is the set of *accepting states*.A Büchi automaton is *deterministic* if for all states s and inputs a, $|\{s' | (s, a, s') \in \delta\}| \leq 1$. Otherwise, it is *nondeterministic*. For a word $w = w_0 w_1 \ldots \in \Sigma^\omega$, a *run* ρ of w is a sequence of states $s_0 s_1 \ldots$ satisfying: (1) $s_0 \in Init$, and (2) $\tau_i = (s_i, w_i, s_{i+1}) \in \delta$ for all i. Let $inf(\rho)$ denote the set of states that occur infinitely often in run ρ. A run ρ is an *accepting run* if $inf(\rho) \cap \mathcal{F} \neq \emptyset$. A word w is an accepting word if it has an accepting run.

The language $\mathcal{L}(\mathcal{A})$ of Büchi automaton $\mathcal{A}$ is the set of all words accepted by it. Büchi automata are known to be closed under set-theoretic union, intersection, and complementation. For Büchi automata A and B, the *language-equivalence* and *language-inclusion* are whether $\mathcal{L}(A) \equiv \mathcal{L}(B)$ and $\mathcal{L}(A) \subseteq \mathcal{L}(B)$, resp.

Let $A = A[0], A[1], \ldots$ be a natural-number sequence, $d > 1$ be a rational number. The *discounted-sum* of A with discount-factor d is $DS(A, d) = \Sigma_{i=0}^{\infty} \frac{A[i]}{d^i}$. For number sequences A and B, (A, B) and $(A - B)$ denote the sequences where the i-th element is $(A[i], B[i])$ and $A[i] - B[i]$, respectively.

Discounted-sum automata A *discounted-sum automaton* with discount-factor $d > 1$, *DS-automaton* in short, is a tuple $\mathcal{A} = (\mathcal{M}, \gamma)$, where $\mathcal{M} = (S, \Sigma, \delta, Init, S)$ is a Büchi automaton, and $\gamma : \delta \to \mathbb{N}$ is the *weight function* that assigns a weight to each transition of automaton $\mathcal{M}$. *Words* and *runs* in weighted ω-automata are defined as they are in Büchi automata. Note that all states are accepting states in this definition. The *weight sequence* of run $\rho = s_0 s_1 \ldots$ of word $w = w_0 w_1 \ldots$ is given by $wt_\rho = n_0 n_1 n_2 \ldots$ where $n_i = \gamma(s_i, w_i, s_{i+1})$ for all i. The *weight of a run* ρ is given by $DS(wt_\rho, d)$. For simplicity, we denote this by $DS(\rho, d)$. The *weight of a word* in DS-automata is defined as $wt_{\mathcal{A}}(w) = sup\{DS(\rho, d) | \rho$ is a run of w in $\mathcal{A}\}$. By convention, if a word $w \notin \mathcal{L}(\mathcal{A})$, then $wt_{\mathcal{A}}(w) = 0$ [15]. A DS-automata is said to be *complete* if from every state there is at least one transition on every alphabet. Formally, for all $p \in S$ and for all $a \in \Sigma$, there exists $q \in S$

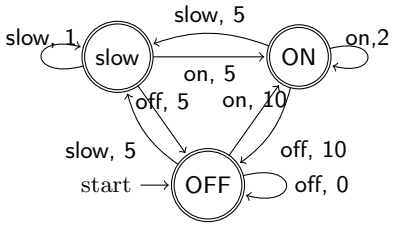


Fig. 1: System S

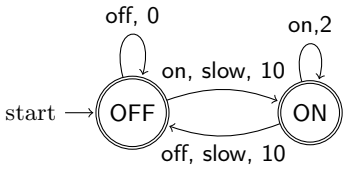


Fig. 2: Specification P

s.t $(p, a, q) \in \delta$. A run $\rho \in P$ of word $w \in \mathcal{L}(P)$ is a *diminished run* if there exists a run $\sigma \in Q$ over the same word w s.t. $DS(\rho, d) < DS(\sigma, d)$. We abuse notation, and use $w \in \mathcal{A}$ to mean $w \in \mathcal{L}(\mathcal{A})$ for Büchi automaton or DS-automaton $\mathcal{A}$. We limit ourselves to integer discount-factors only. Given DS-automata P and Q and discount-factor $d > 1$, the *discounted-sum inclusion problem*, denoted by $P \subseteq_d Q$, determines whether for all words $w \in \Sigma^\omega$, $wt_P(w) \leq wt_Q(w)$.

Comparator automata For natural number μ, integer discount-factor $d > 1$ and inequality relation $\leq$, the *discounted-sum comparator* $\mathcal{A}^{\mu,d}_{\leq}$, *comparator*, in short, is a Büchi automaton that accepts (infinite) words over the alphabet $\{0, 1 \ldots, \mu - 1\} \times \{0, 1 \ldots, \mu - 1\}$ such that a pair (A, B) of sequences is in $\mathcal{L}(\mathcal{A}^{\mu}_{d})$ iff $DS(A, d) \leq DS(B, d)$. Closure properties of Büchi automata ensure that comparator exists for all inequality relations [9].

Motivating example As an example of such a problem formulation, consider the system and specification in Figure 1 and Figure 2, respectively [15]. Here, the specification P depicts the worst-case energy-consumption model for a motor, and the system S is a candidate implementation of the motor. Transitions in S and P are labeled by transition-action and transition-cost. The cost of an execution (a sequence of actions) is given by an *aggregate* of the costs of transitions along its run (a sequence of automaton states). In non-deterministic automata, where each execution may have multiple runs, cost of the execution is the cost of the run with maximum cost. A critical question here is to check whether implementation S is more energy-efficient than specification P. This problem can be framed as a problem of quantitative inclusion between S and P.

3 Prior work

We discuss existing algorithms for DS-inclusion i.e. DetLP and BCV in detail.

3.1 DetLP: DS-determinization and LP-based

Böker and Henzinger studied complexity and decision-procedures for determinization of DS-automata in detail [10]. They proved that a DS-automata can be determinized if it is complete, all its states are accepting states and the discount-factor is an integer. Under all other circumstances, DS-determinization may not be guaranteed. DS-determinization extends subset-construction for automata over finite words. Every state of the determinized DS-automata is represented by an $|S|$-tuple of numbers, where $S = \{q_1, \ldots q_{|S|}\}$ denotes the set of states of

the original DS-automaton. The value stored in the i-th place in the $|S|$-tuple represents the "gap" or extra-cost of reaching state q_i over a finite-word w compared to its best value so far. The crux of the argument lies in proving that when the DS-automata is complete and the discount-factor is an integer, the "gap" can take only finitely-many values, yielding finiteness of the determinized DS-automata, albeit exponentially larger than the original.

Theorem 1. [10] [DS-determinization analysis] *Let A be a complete DS-automata with maximum weight μ over transitions and s number of states. DS-determinization of A generates a DS-automaton with at most μ^s states.*

Chatterjee et al. reduced $P \subseteq_d Q$ between non-deterministic DS-automata P and deterministic DS-automata Q to linear-programming [7,8,15]. First, the product DS-automata $P \times Q$ is constructed so that $(s_P, s_Q) \xrightarrow{a} (t_P, t_Q)$ is a transition with weight $w_P - w_Q$ if transition $s_M \xrightarrow{a} t_M$ with weight w_M is present in M, for $M \in \{P, Q\}$. $P \subseteq_q Q$ is False iff the weight of any word in $P \times Q$ is greater than 0. Since Q is deterministic, it is sufficient to check if the maximum weight of all infinite paths from the initial state in $P \times Q$ is greater than 0. For discounted-sum, the maximum weight of paths from a given state can be determined by a linear-program: Each variable (one for each state) corresponds to the weight of paths originating in this state, and transitions decide the constraints which relate the values of variables (or states) on them. The objective is to maximize weight of variable corresponding to the initial state.

Therefore, the DetLP method for $P \subseteq_d Q$ is as follows: Determinize Q to Q_D via DS-determinization method from [10], and reduce $P \subseteq_d Q_D$ to linear programming following [15]. Note that since determinization is possible only if the DS-automaton is complete, DetLP can be applied only if Q is complete.

Lemma 1. *Let P and Q be non-deterministic DS-automata with s_P and s_Q number of states respectively, τ_P states in P. Let the alphabet be Σ and maximum weight on transitions be μ. Then $P \subseteq_d Q$ is reduced to linear programming with $\mathcal{O}(s_P \cdot \mu^{s_Q})$ variables and $\mathcal{O}(\tau_P \cdot \mu^{s_Q} \cdot |\Sigma|)$ constraints.*

Anderson and Conitzer [7] proved that this system of linear equations can be solved in $\mathcal{O}(m \cdot n^2)$ for m constraints and n variables. Therefore,

Theorem 2. [7,15] [Complexity of DetLP] *Let P and Q be DS-automata with s_P and s_Q number of states respectively, τ_P states in P. Let the alphabet be Σ and maximum weight on transitions be μ. Complexity of DetLP is $\mathcal{O}(s_P^2 \cdot \tau_P \cdot \mu^{s_Q} \cdot |\Sigma|)$.*

3.2 BCV: Comparator-based approach

The key idea behind BCV is that $P \subseteq_d Q$ holds iff every run of P is a diminished run. As a result, BCV constructs an intermediate Büchi automaton *Dim* that consists of all diminished runs of P. It then checks whether *Dim* consists of all runs of P, by determining language-equivalence between *Dim* and an automaton $\hat{P}$ that consists of all runs of P. The comparator $\mathcal{A}^{\mu,d}_{\leq}$ is utilized in the construction of *Dim* to compare weight of runs in P and Q.

1: **Input:** Weighted automata P, Q, and discount-factor d
2: **Output:** True if $P \subseteq_d Q$, False otherwise
3: $\hat{P} \leftarrow \mathsf{AugmentWtAndLabel}(P)$
4: $\hat{Q} \leftarrow \mathsf{AugmentWtAndLabel}(Q)$
5: $\hat{P} \times \hat{Q} \leftarrow \mathsf{MakeProductSameAlpha}(\hat{P}, \hat{Q})$
6: $\mu \leftarrow \mathsf{MaxWeight}(P, Q)$
7: $\mathcal{A}_{\leq}^{\mu,d} \leftarrow \mathsf{MakeComparator}(\mu, d)$
8: $DimWithWitness \leftarrow \mathsf{Intersect}(\hat{P} \times \hat{Q}, \mathcal{A}_{\leq}^{\mu,d})$
9: $Dim \leftarrow \mathsf{FirstProject}(DimWithWitness)$
10: **return** $\hat{P} \equiv Dim$

Algorithm 1: $\mathsf{BCV}(P, Q, d)$, Is $P \subseteq_d Q$?

Strictly speaking, BCV as presented in [9], is a generic algorithm for inclusion under a general class of aggregate functions, called ω-regular aggregate functions. Here, BCV (Algorithm 1) refers to its adaptation to DS. Procedure AugmentWtAndLabel separates between runs of the same word in DS-automata by assigning a unique transition-identity to each transition. It also appends the transition weight, to enable weight comparison afterwards. Specifically, it transforms DS-automaton $\mathcal{A}$ into Büchi automaton $\hat{\mathcal{A}}$, with all states as accepting, by converting transition $\tau = s \xrightarrow{a} t$ with weight wt and unique transition-identity l to transition $\hat{\tau} = s \xrightarrow{(a,wt,l)} t$ in $\hat{\mathcal{A}}$. Procedure $\mathsf{MakeProductSameAlpha}(\hat{P}, \hat{Q})$ takes the product of $\hat{P}$ and $\hat{Q}$ over the same word i.e., transitions $s_{\mathcal{A}} \xrightarrow{(a,n_{\mathcal{A}},l_{\mathcal{A}})} t_{\mathcal{A}}$ in $\mathcal{A}$, for $\mathcal{A} \in \{\hat{P}, \hat{Q}\}$, generates transition $(s_P, s_Q) \xrightarrow{(a,n_P,l_P,n_Q,l_Q)} (t_P, t_Q)$ in $\hat{P} \times \hat{Q}$. The comparator $\mathcal{A}_{\leq}^{\mu,d}$ is constructed with upper-bound μ that equals the maximum weight of transitions in P and Q, and discount-factor d. Intersect matches the alphabet of $\hat{P} \times \hat{Q}$ with $\mathcal{A}_{\leq}^{\mu,d}$, and intersects them. The resulting automaton $DimWithWitness$ accepts word $(w, wt_P, id_P, wt_Q, id_Q)$ iff $DS(wt_P, d) \leq DS(wt_Q, d)$. The projection of $DimWithWitness$ on the first three components of $\hat{P}$ returns Dim which contains the word (w, wt_P, id_P) iff it is a diminished run in P. Finally, language-equivalence between Dim and $\hat{P}$ returns the answer.

Unlike DetLP, BCV operates on incomplete DS-automata as well, and can be extended to DS-automata in which not all states are accepting.

4 QuIP: BCV-based solver for DS-inclusion

We investigate more closely why BCV does not lend itself to a practical implementation for DS-inclusion (§ 4.1). We identify its drawbacks, and propose an improved algorithm QuIP as is described in § 4.3. QuIP improves upon BCV by means of a new optimized comparator that we describe in §4.2.

4.1 Analysis of BCV

The proof for PSPACE-complexity of BCV relies on LI to be PSPACE. In practice, though, implementations of LI apply Ramsey-based inclusion testing [6], rank-based methods [18] etc. All of these algorithms are exponential in time and space

in the worst case. Any implementation of BCV will have to rely on an LI-solver. Therefore, in practice BCV is also exponential in time and space. In fact, we show that its worst-case complexity (in practice) is poorer than DetLP.

Another reason that prevents BCV from practical implementations is that it does not optimize the size of intermediate automata. Specifically, we show that the size and transition-density of *Dim*, which is one of the inputs to LI-solver, is very high (Transition density is the ratio of transitions to states). Both of these parameters are known to be deterrents to the performance of existing LI-solvers [5], subsequently to BCV as well:

Lemma 2. *Let s_P, s_Q, s_d and τ_P, τ_Q, τ_d denote the number of states and transitions in P, Q, and $\mathcal{A}_{\leq}^{\mu,d}$, respectively. Number of states and transitions in Dim are $\mathcal{O}(s_P s_Q s_d)$ and $\mathcal{O}(\tau_P^2 \tau_Q^2 \tau_d |\Sigma|)$, respectively.*

Proof. It is easy to see that the number of states and transitions of $\hat{P}$ $\hat{Q}$ are the same as those of P and Q, respectively. Therefore, the number of states and transitions in $\hat{P} \times \hat{Q}$ are $\mathcal{O}(s_P s_Q)$ and $\mathcal{O}(\tau_P \tau_Q)$, respectively. The alphabet of $\hat{P} \times \hat{Q}$ is of the form $(a, wt_1, id_1, wt_2, id_2)$ for $a \in \Sigma$, wt_1, wt_2 are non-negative weights bounded by μ and id_i are unique transition-ids in P and Q respectively. The alphabet of comparator $\mathcal{A}_{\leq}^{\mu,d}$ is of the form (wt_1, wt_2). To perform intersection of these two, the alphabet of comparator needs to be matched to that of the product, causing a blow-up in number of transitions in the comparator by a factor of $|\Sigma| \cdot \tau_P \cdot \tau_Q$. Therefore, the number of states and transitions in *DimWithWitness* and *Dim* is given by $\mathcal{O}(s_P s_Q s_d)$ and $\mathcal{O}(\tau_P^2 \tau_Q^2 \tau_d |\Sigma|)$.

The comparator is a non-deterministic Büchi automata with $\mathcal{O}(\mu^2)$ states over an alphabet of size μ^2 [9]. Since transition-density $\delta = |S| \cdot |\Sigma|$ for non-deterministic Büchi automata, the transition-density of the comparator is $\mathcal{O}(\mu^4)$. Therefore,

Corollary 1. *Let s_P, s_Q, s_d denote the number of states in P, Q, $\mathcal{A}_{\leq}^{\mu,d}$, respectively, and δ_P, δ_Q and δ_d be their transition-densities. Number of states and transition-density of Dim are $\mathcal{O}(s_P s_Q \mu^2)$ and $\mathcal{O}(\delta_P \delta_Q \tau_P \tau_Q \cdot \mu^4 \cdot |\Sigma|)$, respectively.*

The corollary illustrates that the transition-density of *Dim* is very high even for small inputs. The blow-up in number of transitions of *DimWithWitness* (hence *Dim*) occurs during alphabet-matching for Büchi automata intersection (Algorithm 1, Line 8). However, the blow-up can be avoided by performing intersection over a substring of the alphabet of $\hat{P} \times \hat{Q}$. Specifically, if $s_1 \xrightarrow{(a, n_P, id_P, n_Q, id_Q)} s_2$ and $t_1 \xrightarrow{(wt_1, wt_2)} t_2$ are transitions in $\hat{P} \times \hat{Q}$ and comparator $\mathcal{A}_{\leq}^{\mu,d}$ respectively, then $(s_1, t_1, i) \xrightarrow{(a, n_P, id_P, n_Q, id_Q)} (s_2, t_2, j)$ is a transition in the intersection iff $n_P = wt_1$ and $n_Q = wt_2$, where $j = (i+1) \mod 2$ if either s_1 or t_1 is an accepting state, and $j = i$ otherwise. We call intersection over substring of alphabet IntersectSelectAlpha. The following is easy to prove:

Lemma 3. *Let $\mathcal{A}_1 = \mathsf{Intersect}(\hat{P} \times \hat{Q}, \mathcal{A}_{\leq}^{\mu,d})$, and $\mathcal{A}_2 = \mathsf{IntersectSelectAlpha}(\hat{P} \times \hat{Q}, \mathcal{A}_{\leq}^{\mu,d})$. Intersect extends alphabet of $\mathcal{A}_{\leq}^{\mu,d}$ to match the alphabet of $\hat{P} \times \hat{Q}$ and*

IntersectSelectAlpha *selects a substring of the alphabet of* $\hat{P} \times \hat{Q}$ *as defined above. Then,* $\mathcal{L}(\mathcal{A}_1) \equiv \mathcal{L}(\mathcal{A}_2)$.

IntersectSelectAlpha prevents the blow-up by $|\Sigma| \cdot \tau_P \cdot \tau_Q$, resulting in only $\mathcal{O}(\tau_P \tau_Q \tau_d)$ transitions in *Dim* Therefore,

Lemma 4. [Trans. Den. in BCV] *Let* δ_P, δ_Q *denote transition-densities of* P *and* Q, *resp., and* μ *be the upper bound for comparator* $\mathcal{A}_{\leq}^{\mu,d}$. *Number of states and transition-density of Dim are* $\mathcal{O}(s_P s_Q \mu^2)$ *and* $\mathcal{O}(\delta_P \delta_Q \cdot \mu^4)$, *respectively.*

Language-equivalence is performed via tools for language-inclusion. The most effective tool for language-inclusion RABIT [1] is based on Ramsay-based inclusion testing [6]. The worst-case complexity for $A \subseteq B$ via Ramsay-based inclusion testing is known to be $2^{\mathcal{O}(n^2)}$, when B has n states. Therefore,

Theorem 3. [Practical complexity of BCV] *Let* P *and* Q *be DS-automata with* s_P, s_Q *number of states respectively, and maximum weight on transitions be* μ. *Worst-case complexity for* BCV *for integer discount-factor* $d > 1$ *when language-equivalence is performed via Ramsay-based inclusion testing is* $2^{\mathcal{O}(s_P^2 \cdot s_Q^2 \cdot \mu^4)}$.

Recall that language-inclusion queries are $\hat{P} \subseteq Dim$ and $Dim \subseteq \hat{P}$. Since *Dim* has many more states than $\hat{P}$, the complexity of $\hat{P} \subseteq Dim$ dominates.

Theorem 2 and Theorem 3 demonstrate that the complexity of BCV (in practice) is worse than DetLP.

4.2 Baseline automata: An optimized comparator

The $2^{\mathcal{O}(s^2)}$ dependence of BCV on the number of states s of the comparator motivates us to construct a more compact comparator. Currently a comparator consists of $\mathcal{O}(\mu^2)$ number of states for upper bound μ [9]. In this section, we introduce the related concept of *baseline automata* which consists of only $\mathcal{O}(\mu)$-many states and has transition density of $\mathcal{O}(\mu^2)$.

Definition 1 (Baseline automata). *For natural number* μ, *integer discount-factor* $d > 1$ *and relation* R, *for* $R \in \{\leq, \geq, <, >, =\}$, *the* DSbaseline automata $\mathcal{B}_R^{\mu,d}$, *baseline in short, is a Büchi automaton that accepts (infinite) words over the alphabet* $\{-(\mu-1), \ldots, \mu-1\}$ *s.t. sequences* $V \in \mathcal{L}(\mathcal{B}_R^{\mu,d})$ *iff* $DS(V,d)$ R 0.

Semantically, a baseline automata with upper bound μ, discount-factor d and inequality relation R is the language of all integer sequences bounded by μ for which their discounted-sum is related to 0 by the relation R. Baseline automata can also be said to be related to *cut-point languages* [14].

Since $DS(A,d) \leq DS(B,d) = DS(A-B,d) \leq 0$, $\mathcal{A}_{\leq}^{\mu,d}$ accepts (A,B) iff $\mathcal{B}_{\leq}^{\mu,d}$ accepts $(A-B)$, regularity of baseline automata follows straight-away from the regularity of comparator. In fact, the automaton for $\mathcal{B}_{\leq}^{\mu,d}$ can be derived from $\mathcal{A}_{\leq}^{\mu,d}$ by transforming the alphabet from (a,b) to $(a-b)$ along every transition. The first benefit of the modified alphabet is that its size is reduced from μ^2 to

$2 \cdot \mu - 1$. In addition, it coalesces all transitions between any two states over alphabet $(a, a + v)$, for all a, into one single transition over v, thereby also reducing transitions. However, this direct transformation results in a baseline with $\mathcal{O}(\mu^2)$ states. We provide a construction of baseline with $\mathcal{O}(\mu)$ states only.

The key idea behind the construction of the baseline is that the discounted-sum of sequence V can be treated as a number in base d i.e. $DS(V, d) = \Sigma_{i=0}^{\infty} \frac{V[i]}{d^i} = (V[0].V[1]V[2]\dots)_d$. So, there exists a non-negative value C in base d s.t. $V + C = 0$ for arithmetic operations in base d. This value C can be represented by a non-negative sequence C s.t. $DS(C, d) + DS(V, d) = 0$. Arithmetic in base d over sequences C and V result in a sequence of carry-on X such that:

Lemma 5. *Let V, C, X be the number sequences, $d > 1$ be a positive integer such that following equations holds true:*

1. *When $i = 0$, $V[0] + C[0] + X[0] = 0$*
2. *When $i \geq 1$, $V[i] + C[i] + X[i] = d \cdot X[i-1]$*

Then $DS(V, d) + DS(C, d) = 0$.

In the construction of the comparator, it has been proven that when A and B are bounded non-negative integer sequences s.t. $DS(A, d) \leq DS(B, d)$, the corresponding sequences C and X are also bounded integer-sequences [9]. The same argument transcends here: When V is a bounded integer sequence s.t. $DS(V, d) \leq 0$, there exists a corresponding pair of bounded integer sequence C and X. In fact, the bounds used for the comparator carry over to this case as well. Sequence C is non-negative and is bounded by $\mu_C = \mu \cdot \frac{d}{d-1}$ since $-\mu_C$ is the minimum value of discounted-sum of V, and integer-sequence X is bounded by $\mu_X = 1 + \frac{\mu}{d-1}$. On combining Lemma 5 with the bounds on X and C we get:

Lemma 6. *Let V and be an integer-sequence bounded by μ s.t. $DS(V, d) \leq 0$, and X be an integer sequence bounded by $(1 + \frac{\mu}{d-1})$, then there exists an X s.t.*

1. *When $i = 0$, $0 \leq -(X[0] + V[0]) \leq \mu \cdot \frac{d}{d-1}$*
2. *When $i \geq 1$, $0 \leq (d \cdot X[i-1] - V[i] - X[i]) \leq \mu \cdot \frac{d}{d-1}$*

Equations 1-2 from Lemma 6 have been obtained by expressing $C[i]$ in terms of $X[i], X[i-1], V[i]$ and d, and imposing the non-negative bound of $\mu_C = \mu \cdot \frac{d}{d-1}$ on the resulting expression. Therefore, Lemma 6 implicitly captures the conditions on C by expressing it only in terms of V, X and d for $DS(V, d) \leq 0$ to hold.

In construction of the baseline automata, the values of $V[i]$ is part of the alphabet, upper bound μ and discount-factor d are the input parameters. The only unknowns are the value of $X[i]$. However, we know that it can take only finitely many values i.e. integer values $|X[i]| \leq \mu_X$. So, we store all possible values of $X[i]$ in the states. Hence, the state-space S comprises of $\{(x) || x| \leq \mu_X\}$ and a start state s. Transitions between these states are possible iff the corresponding x-values and alphabet v satisfy the conditions of Equations 1-2 from Lemma 6. There is a transition from start state s to state (x) on alphabet

v if $0 \leq -(x+v) \leq \mu \cdot \frac{d}{d-1}$, and from state (x) to state (x') on alphabet v if $0 \leq (d \cdot x - v - x') \leq \mu \cdot \frac{d}{d-1}$. All (x)-states are accepting. This completes the construction for baseline automaton $\mathcal{B}_{\leq}^{\mu,d}$. Clearly $\mathcal{B}_{\leq}^{\mu,d}$ has only $\mathcal{O}(\mu)$ states.

Since Büchi automata are closed under set-theoretic operations, baseline automata is ω-regular for all other inequalities too. Moreover, baseline automata for all other inequalities also have $\mathcal{O}(\mu)$ states. Therefore for sake of completion, we extend $\mathcal{B}_{\leq}^{\mu,d}$ to construct $\mathcal{B}_{<}^{\mu,d}$. For $DS(V,d) < 0$, $DS(C,d) > 0$ (implicitly generated C). Since C is a non-negative sequence it is sufficient if at least one value of C is non-zero. Therefore, all runs are diverted to non-accepting states $(x, \perp)$ using the same transitions until the value of c is zero, and moves to accepting states (x) only if it witnesses a non-zero value for c. Formally,

Construction Let $\mu_C = \mu \cdot \frac{d}{d-1} \leq 2 \cdot \mu$ and $\mu_X = 1 + \frac{\mu}{d-1}$. $\mathcal{B}_{<}^{\mu,d} = (S, \Sigma, \delta_d, \mathit{Init}, \mathcal{F})$

- $S = \mathit{Init} \cup \mathcal{F} \cup S_\perp$ where
  $\mathit{Init} = \{s\}$, $\mathcal{F} = \{x || x| \leq \mu_X\}$, and
  $S_\perp = \{(x, \perp) || x| \leq \mu_X\}$ where $\perp$ is a special character, and $x \in \mathbb{Z}$.
- $\Sigma = \{v : |v| \leq \mu\}$ where v is an integer.
- $\delta_d \subset S \times \Sigma \times S$ is defined as follows:
  1. Transitions from start state s:
     i (s, v, x) for all $x \in \mathcal{F}$ s.t. $0 < -(x+v) \leq \mu_C$
     ii $(s, v, (x, \perp))$ for all $(x, \perp) \in S_\perp$ s.t. $x + v = 0$
  2. Transitions within $S_\perp$: $((x, \perp), v, (x', \perp))$ for all $(x, \perp)$, $(x', \perp) \in S_\perp$, if $d \cdot x = v + x'$
  3. Transitions within $\mathcal{F}$: (x, v, x') for all $x, x' \in \mathcal{F}$ if $0 \leq d \cdot x - v - x' < d$
  4. Transition between $S_\perp$ and $\mathcal{F}$: $((x, \perp), v, x')$ for $(x, \perp) \in S_\perp$, $x' \in \mathcal{F}$ if $0 < d \cdot x - v - x' < d$

Theorem 4. [Baseline] *The Büchi automaton constructed above is the baseline $\mathcal{B}_{<}^{\mu,d}$ with upper bound μ, integer discount-factor $d > 1$ and relation $<$.*

The baseline automata for all inequality relations will have $\mathcal{O}(\mu)$ states, alphabet size of $2 \cdot \mu - 1$, and transition-density of $\mathcal{O}(\mu^2)$.

4.3 QuIP: Algorithm description

The construction of the universal leads to an implementation-friendly QuIP from BCV. The core focus of QuIP is to ensure that the size of intermediate automata is small and they have fewer transitions to assist the LI-solvers. Technically, QuIP differs from BCV by incorporating the baseline automata and an appropriate IntersectSelectAlpha function, rendering QuIP theoretical improvement over BCV. Like BCV, QuIP also determines all diminished runs of P. So, it disambiguates P by appending weight and a unique label to each of its transitions. Since, the identity of runs of Q is not important, we do not disambiguate between runs of Q, we only append the weight to each transition (Algorithm 2, Line 4). The baseline automaton is constructed for discount-factor d, maximum weight

1: **Input:** Weighted automata P, Q, and discount-factor d
2: **Output:** True if $P \subseteq_d Q$, False otherwise
3: $\hat{P} \leftarrow \mathsf{AugmentWtAndLabel}(P)$
4: $\hat{Q} \leftarrow \mathsf{AugmentWt}(Q)$
5: $\hat{P} \times \hat{Q} \leftarrow \mathsf{MakeProductSameAlpha}(\hat{P}, \hat{Q})$
6: $\mathcal{A} \leftarrow \mathsf{MakeBaseline}(\mu, d, \leq)$
7: $\mathit{DimWithWitness} \leftarrow \mathsf{IntersectSelectAlpha}(\hat{P} \times \hat{Q}, \mathcal{A})$
8: $\mathit{Dim} \leftarrow \mathsf{ProjectOutWt}(\mathit{DimWithWitness})$
9: $\hat{P}_{-wt} \leftarrow \mathsf{ProjectOutWt}(\hat{P})$
10: **return** $\hat{P}_{-wt} \subseteq \mathit{Dim}$

Algorithm 2: $\mathsf{QuIP}(P, Q, d)$, Is $P \subseteq_d Q$?

μ along transitions in P and Q, and the inequality $\leq$. Since the alphabet of the baseline automata are integers between $-\mu$ to μ, the alphabet of the product $\hat{P} \times \hat{\mathbb{Q}}$ is adjusted accordingly. Specifically, the weight recorded along transitions in the product is taken to be the difference of weight in $\hat{P}$ to that in $\hat{Q}$ i.e. if $\tau_P : s_1 \xrightarrow{a_1, wt_1, l} s_2$ and $\tau_Q : t_1 \xrightarrow{a_2, wt_2} t_2$ are transitions in $\hat{P}$ and $\hat{Q}$ respectively, then $\tau = (s_1, t_1) \xrightarrow{a_1, wt_1 - wt_2, l} (s_2, t_2)$ is a transition in $\hat{P} \times \hat{Q}$ iff $a_1 = a_2$ (Algorithm 2, Line 5). In this case, $\mathsf{IntersectSelectAlpha}$ intersects baseline automata $\mathcal{A}$ and product $\hat{P} \times \hat{Q}$ only on the weight-component of alphabet in $\hat{P} \times \hat{Q}$. Specifically, if $s_1 \xrightarrow{(a, wt_1, l)} s_2$ and $t_1 \xrightarrow{wt_2} t_2$ are transitions in $\hat{P} \times \hat{Q}$ and comparator $\mathcal{A}^{\mu,d}_{\leq}$ respectively, then $(s_1, t_1, i) \xrightarrow{a, wt_1, l} (s_2, t_2, j)$ is a transition in the intersection iff $wt_1 = wt_2$, where $j = (i + 1) \mod 2$ if either s_1 or t_1 is an accepting state, and $j = i$ otherwise. Automaton Dim and $\hat{P}_{-wt}$ are obtained by project out the weight-component from the alphabet of $\hat{P} \times \hat{Q}$ and $\hat{P}$ respectively. The alphabet of $\hat{P} \times \hat{Q}$ and $\hat{P}$ are converted from (a, wt, l) to only (a, l). It is necessary to project out the weight component since in $\hat{P} \times \hat{Q}$ they represent the difference of weights and and in $\hat{P}$ they represent the absolute value of weight.

Finally, the language of Dim is equated with that of $\hat{P}_{-wt}$ which is the automaton generated from $\hat{P}$ after discarding weights from transitions. However, it is easy to prove that $\mathit{Dim} \subseteq \hat{P}_{-wt}$. Therefore, instead of language-equivalence between Dim and $\hat{P}_{-wt}$ and, it is sufficient to check whether $\hat{P}_{-wt} \subseteq \mathit{Dim}$. As a result, QuIP utilizes LI-solvers as a black-box to perform this final step.

Lemma 7. [Trans. Den. in QuIP] *Let δ_P, δ_Q denote transition-densities of P and Q, resp., and μ be the upper bound for baseline $\mathcal{B}^{\mu,d}_{\leq}$. Number of states and transition-density of Dim are $\mathcal{O}(s_P s_Q \mu)$ and $\mathcal{O}(\delta_P \delta_Q \cdot \mu^2)$, respectively.*

Theorem 5. [Practical complexity of QuIP] *Let P and Q be DS-automata with s_P, s_Q number of states, respectively, and maximum weight on transitions be μ. Worst-case complexity for QuIP for integer discount-factor $d > 1$ when language-equivalence is performed via Ramsay-based inclusion testing is $2^{\mathcal{O}(s_P^2 \cdot s_Q^2 \cdot \mu^2)}$.*

Theorem 5 demonstrates that while complexity of QuIP (in practice) improves upon BCV (in practice), it is still worse than DetLP.

5 Experimental evaluation

We provide implementations of our tools QuIP and DetLP and conduct experiments on a large number of synthetically-generated benchmarks to compare their performance. We seek to find answers to the following questions: (1). Which tool has better performance, as measured by runtime, and number of benchmarks solved? (2). How does change in transition-density affect performance of the tools? (3). How dependent are our tools on their underlying solvers?

5.1 Implementation details

We implement our tools QuIP and DetLP in C++, with compiler optimization `o3` enabled. We implement our own library for all Büchi-automata and DS-automata operations, except for language-inclusion for which we use the state-of-the-art LI-solver RABIT [4] as a black-box. We enable the `-fast` flag in RABIT, and tune its JAVA-threads with Xss, Xms, Xmx set to 1GB, 1GB and 8GB respectively. We use the large-scale LP-solver GLPSOL provided by GLPK (GNU Linear Programming Kit) [2] inside DetLP. We did not tune GLPSOL since it consumes a very small percentage of total time in DetLP, as we see later in Fig 4b.

We also employ some implementation-level optimizations. Various steps of QuIP and DetLP such as product, DS-determinization, baseline construction, involve the creation of new automaton states and transitions. We reduce their size by adding a new state only if it is reachable from the initial state, and a new transition only if it originates from such a state.

The universal automata is constructed on the restricted alphabet of only those weights that appear in the product $\hat{P} \times \hat{Q}$ to include only necessary transitions. We also reduce its size with Büchi minimization tool Reduce [4].

Since all states of $\hat{P} \times \hat{Q}$ are accepting, we conduct the intersection so that it avoids doubling the number of product states. This can be done, since it is sufficient to keep track of whether words visit accepting states in the universal.

5.2 Benchmarks

To the best of our knowledge, there are no standardized benchmarks for DS-automata. We attempted to experimented with examples that appear in research papers. However, these examples are too few and too small, and do not render an informative view of performance of the tools. Following a standard approach to performance evaluation of automata-theoretic tools [5,19,22], we experiment with our tools on *randomly generated* benchmarks.

Random weighted-automata generation The parameters for our random weighted-automata generation procedure are the number of states N, transition-density δ and upper-bound μ for weight on transitions. The states are represented by the set $\{0, 1, \ldots, N-1\}$. All states of the weighted-automata are accepting, and they have a unique initial state 0. The alphabet for all weighted-automata is fixed to $\Sigma = \{a, b\}$. Weight on transitions ranges from 0 to $\mu - 1$. For our

experiments we only generate complete weighted-automata. These weighted automata are generated only if the number of transitions $\lfloor N \cdot \delta \rfloor$ is greater than $N \cdot |\Sigma|$, since there must be at least one transition on each alphabet from every state. We first complete the weighted-automata by creating a transition from each state on every alphabet. In this case the destination state and weight are chosen randomly. The remaining $(N \cdot |\Sigma| - \lfloor N \cdot \delta \rfloor)$-many transitions are generated by selecting all parameters randomly i.e. the source and destination states from $\{0, \ldots N-1\}$, the alphabet from Σ, and weight on transition from $\{0, \mu - 1\}$.

5.3 Design and setup for experimental evaluation

Our experiments were designed with the objective to compare DetLP and QuIP. Due to the lack of standardized benchmarks, we conduct our experiments on randomly-generated benchmarks. Therefore, the parameters for $P \subseteq_d Q$ are the number of states s_P and s_Q, transition density δ, and maximum weight wt. We seek to find answers to the questions described at the beginning of § 5.

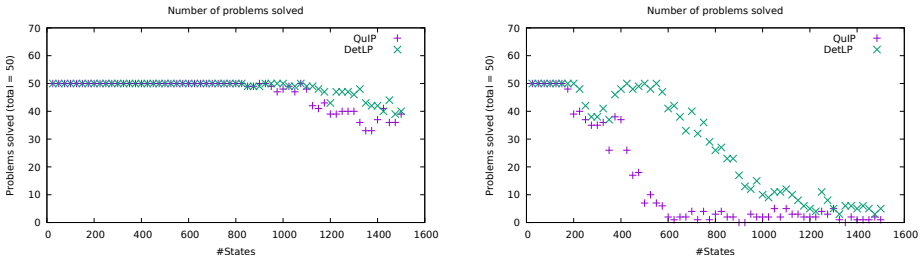
Each instantiation of the parameter-tuple (s_P, s_Q, δ, wt) and a choice of tool between QuIP and DetLP corresponds to one experiment. In each experiment, the weighted-automata P and Q are randomly-generated with the parameters (s_P, δ, wt) and (s_Q, δ, wt), respectively, and language-inclusion is performed by the chosen tool. Since all inputs are randomly-generated, each experiment is repeated for 50 times to obtain statistically significant data. Each experiment is run for a total of 1000 sec on for a single node of a high-performance cluster. Each node of the cluster consists of two quad-core Intel-Xeon processor running at 2.83GHz, with 8GB of memory per node. The runtime of experiments that do not terminate within the given time limit is assigned a runtime of ∞. We report the median of the runtime-data collected from all iterations of the experiment.

These experiments are scaled-up by increasing the size of inputs. The worst-case analysis of QuIP demonstrates that it is symmetric in s_P and s_Q, making the algorithm impartial to which of the two inputs is scaled (Theorem 5). On the other hand, complexity of DetLP is dominated by s_Q (Theorem 2). Therefore, we scale-up our experiments by increasing s_Q only.

Since DetLP is restricted to complete automata, these experiments are conducted on complete weighted automata only. We collect data on total runtime of each tool, the time consumed by the underlying solver, and the number of times each experiment terminates with the given resources. We experiment with $s_P = 10$, δ ranges between 2.5-4 in increments of 0.5 (we take lower-bound of 2.5 since $|\Sigma| = 2$), $wt \in \{4, 5\}$, and s_Q ranges from 0-1500 in increments of 25, $d = 3$. These sets of experiments also suffice for testing scalability of both tools.

5.4 Observations

We first compare the tools based on the number of benchmarks each can solve. We also attempt to unravel the main cause of failure of each tool. Out of the 50 experiments for each parameter-value, DetLP consistently solves more bench-



(a) (b)

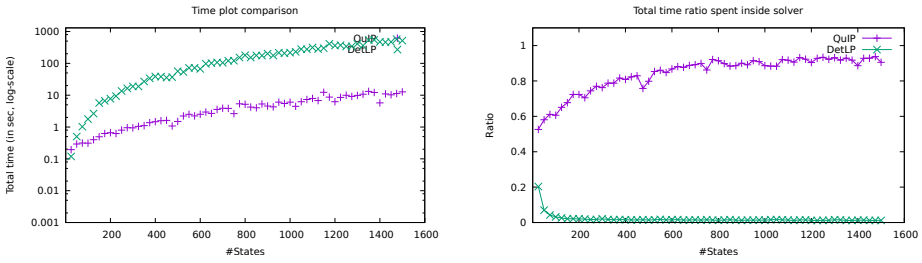
Fig. 3: Number of benchmarks solved out of 50 as s_Q increases with $s_P = 10$, $\mu = 4$. $\delta = 2.5$ and $\delta = 4$ in Fig 3a and Fig 3b, respectively.

marks than QuIP for the same parameter-values (Fig. 3a-3b)[1]. The figures also reveal that both tools solve more benchmarks at lower transition-density. The most common, in fact almost always, reason for QuIP to fail before its timeout was reported to be memory-overflow inside RABIT during language-inclusion between $\hat{P}_{-wt}$ and Dim. On the other hand, the main cause of failure of DetLP was reported to be memory overflow during DS-determinization and preprocessing of the determinized DS-automata before GLPSOL is invoked. This occurs due to the sheer size of the determinized DS-automata, which can very quickly become very large. These empirical observations indicate that the bottleneck in QuIP and DetLP may be language-inclusion and explicit DS-determinization, respectively.

We investigate the above intuition by analyzing the runtime trends for both tools. Fig. 4a plots the runtime for both tools. The plot shows that QuIP fares significantly better than DetLP in runtime at $\delta = 2.5$. The plots for both the tools on logscale seem curved (Fig. 4a), suggesting a sub-exponential runtime complexity. These were observed at higher δ as well. However, at higher δ we observe very few outliers on the runtime-trend graphs of QuIP at larger inputs when just a few more than 50% of the runs are successful. This is expected since effectively, the median reports the runtime of the slower runs in these cases. Fig 4b records the ratio of total time spent inside RABIT and GLPSOL . The plot reveals that QuIP spends most of its time inside RABIT. We also observe that most memory consumptions in QuIP occurs inside RABIT. In contrast, GLPSOL consumes a negligible amount of time and memory in DetLP. Clearly, performance of QuIP and DetLP is dominated by RABIT and explicit DS-determinization, respectively. We also determined how runtime performance of tools changes with increasing discount-factor d. Both tools consume lesser time as d increases.

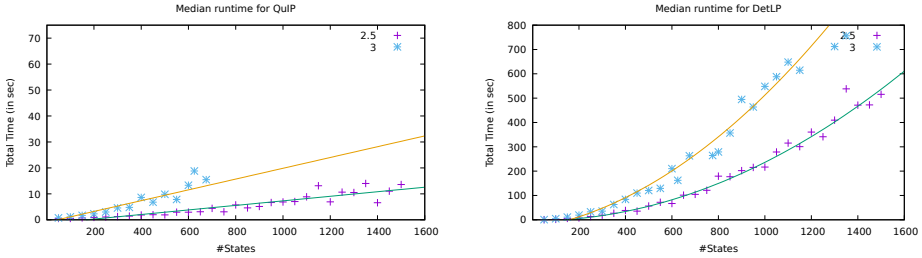
Finally, we test for scalability of both tools. In Fig. 5a, we plot the median of total runtime as s_Q increases at $\delta = 2.5, 3$ ($s_P = 10, \mu = 4$) for QuIP. We attempt to best-fit the data-points for each δ with functions that are linear, quadratic and cubic in s_Q using squares of residuals method. Fig 5b does the same for DetLP. We observe that QuIP and DetLP are best fit by functions that are linear and quadratic in s_Q, respectively.

[1] Figures are best viewed online and in color



(a) (b)

Fig. 4: Time trends: Fig 4a plots total runtime as s_Q increases $s_P = 10$,$\mu = 4$, $\delta = 2.5$. Figure shows median-time for each parameter-value. Fig 4b plots the ratio of time spent by tool inside its solver at the same parameter values.



(a) (b)

Fig. 5: Scalability of QuIP (Fig 5a) and DetLP (Fig 5b) at $\delta = 2.5, 3$. Figures show median-time for each parameter-value.

Inferences and discussion Our empirical analysis arrives at conclusions that a purely theoretical exploration would not have. First of all, we observe that despite having a the worse theoretical complexity, the median-time complexity of QuIP is better than DetLP by an order of n. In theory, QuIP scales exponentially in s_Q, but only linearly in s_Q in runtime. Similarly, runtime of DetLP scales quadratically in s_Q. The huge margin of complexity difference emphasizes why solely theoretical analysis of algorithms is not sufficient.

Earlier empirical analysis of LI-solvers had made us aware of their dependence on transition-density δ. As a result, we were able to design QuIP cognizant of parameter δ. Therefore, its runtime dependence on δ is not surprising. However, our empirical analysis reveals runtime dependence of DetLP on δ. This is unexpected since δ does not appear in any complexity-theoretic analysis of DetLP (Theorem 1). We suspect this behavior occurs because the creation of each transition, say on alphabet a, during DS-determinization requires the procedure to analyze every transition on alphabet a in the original DS-automata. Higher the transition-density, more the transitions in the original DS-automata, hence more expensive is the creation of transitions during DS-determinization.

We have already noted that the performance of QuIP is dominated by RABIT in space and time. Currently, RABIT is implemented in Java. Although RABIT surpasses all other LI-solvers in overall performance, we believe it can

be improved significantly via a more space-efficient implementation in a more performance-oriented language like C++. This would, in-turn, enhance QuIP.

The current implementation of DetLP utilizes the vanilla algorithm for DS-determinization. Since DS-determinization dominates DetLP, there is certainly merit in designing efficient algorithms for DS-determinization. However, we suspect this will be of limited advantage to DetLP since it will persist to incur the complete cost of explicit DS-determinization due to the separation of automata-theoretic and numeric reasoning.

Based on our observations, we propose to extract the complementary strengths of both tools: First, apply QuIP with a small timeout; Since DetLP solves more benchmarks, apply DetLP only if QuIP fails.

6 Concluding remarks and future directions

This paper presents the first empirical evaluation of algorithms and tools for DS-inclusion. We present two tools DetLP and QuIP. Our first tool DetLP is based on explicit DS-determinization and linear programming, and renders an exponential time and space algorithm. Our second tool QuIP improves upon a previously known comparator-based automata-theoretic algorithm BCV by means of an optimized comparator construction, called universal automata. Despite its PSPACE-complete theoretical complexity, we note that all practical implementations of QuIP are also exponential in time and space.

The focus of this work is to investigate these tools in practice. In theory, the exponential complexity of QuIP is worse than DetLP. Our empirical evaluation reveals the opposite: The median-time complexity of QuIP is better than DetLP by an order of n. Specifically, QuIP scales linearly while DetLP scales quadratically in the size of inputs. This re-asserts the gap between theory and practice, and aserts the need of better metrics for practical algorithms. Further emprirical analysis by by scaling the right-hand side automaton will be beneficial.

Nevertheless, DetLP consistently solves more benchmarks than QuIP. Most of QuIP's experiments fail due to memory-overflow within the LI-solver, indicating that more space-efficient implementations of LI-solvers would boost QuIP's performance. We are less optimistic about DetLP though. Our evaluation highlights the impediment of explicit DS-determinization, a cost that is unavoidable in DetLP's separation-of-concerns approach. This motivates future research that integrates automata-theoretic and numerical reasoning by perhaps combining implicit DS-determinzation with baseline automata-like reasoning to design an on-the-fly algorithm for DS-inclusion.

Last but not the least, our empirical evaluations lead to discovering dependence of runtime of algorithms on parameters that had not featured in their worst-case theoretical analysis, such as the dependence of DetLP on transition-density. Such evaluations build deeper understanding of algorithms, and will hopefully serve a guiding light for theoretical and empirical investigation in-tandem of algorithms for quantitative analysis

Acknowledgements. We thank anonymous reviewers for their comments. We thank K. S. Meel, A. A. Shrotri, L. M. Tabajara, and S. Zhu for helpful discussions. This work was partially supported by NSF Grant No. 1704883, "Formal Analysis and Synthesis of Multiagent Systems with Incentives".

References

1. RABIT: Ramsey-based büchi automata inclusion testing.
2. GLPK. `https://www.gnu.org/software/glpk/`.
3. GOAL. `http://goal.im.ntu.edu.tw/wiki/`.
4. Rabit-Reduce. `http://www.languageinclusion.org/`.
5. P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Simulation subsumption in ramsey-based büchi automata universality and inclusion testing. In *Proc. of CAV*, pages 132–147. Springer, 2010.
6. P. A. Abdulla, Y.-F. Chen, L. Clemente, L. Holík, C.-D. Hong, R. Mayr, and T. Vojnar. Advanced ramsey-based büchi automata inclusion testing. In *Proc. of CONCUR*, volume 11, pages 187–202. Springer, 2011.
7. G. Andersen and V. Conitzer. Fast equilibrium computation for infinitely repeated games. In *Proc. of AAAI*, pages 53–59, 2013.
8. D. Andersson. An improved algorithm for discounted payoff games. In *ESSLLI Student Session*, pages 91–98, 2006.
9. S. Bansal, S. Chaudhuri, and M. Y. Vardi. Comparator automata in quantitative verification. In *Proc. of FoSSaCS*, 2018.
10. U. Boker and T. A. Henzinger. Exact and approximate determinization of discounted-sum automata. *LMCS*, 10(1), 2014.
11. U. Boker, T. A. Henzinger, and J. Otop. The target discounted-sum problem. In *Proc. of LICS*, pages 750–761, 2015.
12. P. Cernỳ, K. Chatterjee, T. A. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. In *Proc. of CAV*, volume 6806, pages 243–259. Springer, 2011.
13. P. Cerny, T. A. Henzinger, and A. Radhakrishna. Quantitative abstraction refinement. *ACM SIGPLAN Notices*, 48(1):115–128, 2013.
14. K. Chatterjee, L. Doyen, and T. A. Henzinger. Expressiveness and closure properties for quantitative languages. In *Proc. of LICS*, pages 199–208. IEEE, 2009.
15. K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. *Transactions on Computational Logic*, 11(4):23, 2010.
16. K. Chatterjee, R. Majumdar, and T. A. Henzinger. Markov decision processes with multiple objectives. In *STACS*, volume 6, pages 325–336. Springer, 2006.
17. M. Droste, W. Kuich, and H. Vogler. *Handbook of weighted automata*. Springer, 2009.
18. O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *Transactions on Computational Logic*, 2(3):408–429, 2001.
19. R. Mayr and L. Clemente. Advanced automata minimization. *ACM SIGPLAN Notices*, 48(1):63–74, 2013.
20. A. Rubinstein. Finite automata play the repeated prisoner's dilemma. *Journal of Economic Theory*, 39(1):83–96, 1986.
21. W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
22. D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In *Proc. of LPAR*, pages 396–411. Springer, 2005.

23. W. Thomas, T. Wilke, et al. *Automata, logics, and infinite games: A guide to current research*, volume 2500. Springer Science & Business Media, 2002.