Polynomial Time Decidability of Weighted Synchronization under Partial Observability

Jan Křetínský[1], Kim Guldstrand Larsen[2], Simon Laursen[2], and Jiří Srba[2]

1 **IST Austria**
2 **Aalborg University, Department of Computer Science, Denmark**

Abstract

We consider weighted automata with both positive and negative integer weights on edges and study the problem of synchronization using adaptive strategies that may only observe whether the current weight-level is negative or nonnegative. We show that the synchronization problem is decidable in polynomial time for deterministic weighted automata.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases weighted automata, partial observability, synchronization, complexity

1 Introduction

The problem of synchronizing automata [4, 14, 16] studies the following natural question: *"how can we gain control over a device when its current state is unknown?"* Synchronizing automata have classically been studied in the setting of deterministic finite automata (DFA), aiming at finding short synchronizing words, i.e. finite sequences of input symbols that will bring the automaton from any (unknown) state into a unique state. Here the existence of a synchronizing word is NLOGSPACE-complete [4, 16], and polynomial bounds were given on the length of the shortest synchronizing word. Yet, establishing a tight bound on the length of the shortest synchronizing word has been an open problem for the last 50 years, with Černý [4] conjecturing that words of length at most $(n-1)^2$, where n is the number of states in the DFA, are sufficient.

We consider synchronization of deterministic weighted automata (WA), where their states are composed of locations and integer weights, and where transitions have their associated weights from $\mathbb{Z}$. In this setting, weights are simply accumulated during the run of the system, and thus it is impossible to find a word that will ensure synchronization to a single state: for any two states with identical locations but different weights, e.g. (ℓ, z) and $(\ell, z+1)$, any word will—by the assumption of determinism—maintain the relative difference in their weights. We therefore assume that during the synchronization, the controller has some (minimal) information available concerning the current weight of the system; in particular, we assume that the controller is able to observe whether the current weight is negative or nonnegative. Under this assumption, a solution to the synchronization problem becomes an *adaptive* strategy, in the sense that the next input to be selected may be based on the previous weight-observations made by the controller.

Our main result is that the existence of a synchronizing strategy, using only observations of the sign of the current weight-level, is decidable in polynomial time for deterministic WA. This result relies on a polynomial time algorithm for detecting cycles of weight $+1$ and -1 in a given weighted graph.

Fig. 1 illustrates BP (Blind Packman), a WA with 6 locations and 4 actions. We have to find a strategy that will (under partial observability of weights) synchronize infinitely many

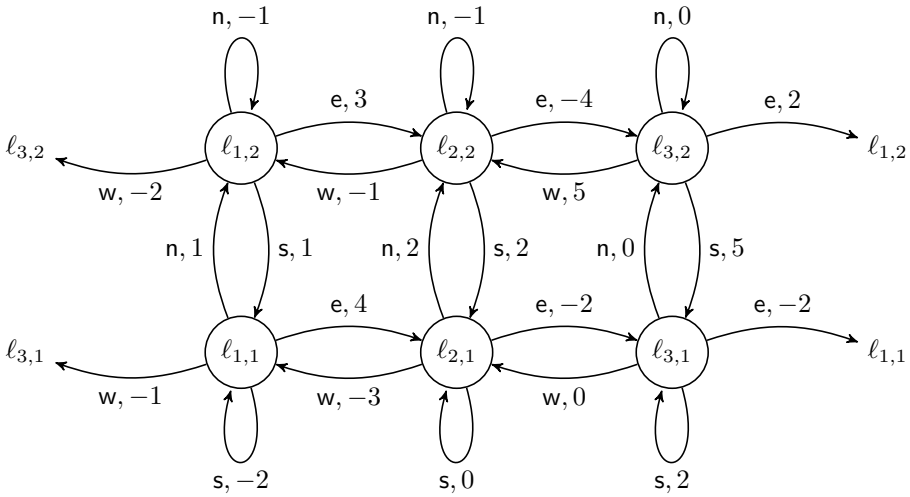

Figure 1 **B**lind **P**ackman with moves north (n), south (s), east (e) and west (w)

states of the form $(\ell_{i,j}, z)$ where $\ell_{i,j}$ is one of the 6 locations and $z \in \mathbb{Z}$ is the starting weight. First, we note that after an n-input, BP will be in one of the 3 (top-row) locations $\ell_{i,2}$ for $i = 1, 2, 3$. Given the cyclic, horizontal structure, it is also clear that no further sequence of inputs from the set $\{\mathsf{n}, \mathsf{e}, \mathsf{s}, \mathsf{w}\}$ can provide any additional information in which of the three locations we are located. However, assuming the weight-level is observed to be nonnegative (a similar case applies if the weight-level is observed to be negative), we may infer that BP is in a state of the form $(\ell_{i,2}, z)$ with $z \geq 0$. Noting the -1-loops from $\ell_{1,2}$ and $\ell_{2,2}$, it is tempting to repeatedly offer n as input until the weight-level becomes negative. However, the presence of the 0-loop at $\ell_{3,2}$ makes it possible that such a strategy will not terminate. Instead, we observe that the input-word $(\mathsf{n} \cdot \mathsf{e})^3$ will constitute a (composite) cycle in BP that makes the weight-level drop exactly by -1, regardless as to from which of the three top-locations the word was executed. Thus, by repeating this input-word while constantly observing the sign of the weight-level and terminating as soon as the weight-level becomes negative, we are able to infer that the BP automaton is either in the location $\ell_{3,2}$ in case the observation changed after the input e, or in one of the states $(\ell_{1,2}, -1)$ or $(\ell_{2,2}, -1)$ if the change happened after the input n. In the former case, we exercise the cycle e^3 until a change in observation brings us to $(\ell_{3,2}, 0)$. In the latter case, an e-input followed by a test of the weight-level will reveal the true identity of the state; using ± 1 cycles, it is now easy to reach $(\ell_{3,2}, 0)$, thus completing the synchronization.

As illustrated by the above example, the presence of cycles with weights $+1$ and -1 is essential for the synchronization under partial observability. As we shall demonstrate, the existence of such cycles is decidable in polynomial time, and constitutes, with a few other polynomial time checks, a necessary and sufficient condition for the synchronization of a WA.

Related work

Survey of results and applications for classical synchronizing words may be found in [14, 16]. Recently, there has been an increasing interest in novel extensions of the synchronization problem. Volkov [9] studied synchronization games and priced synchronization on weighted automata with positive weights and finds a synchronizing word where the worst accumulated

cost is below a given bound. In [6, 7, 8], (infinite) synchronizing words were studied in the probabilistic settings. Synchronization of weighted timed automata was studied in [5], where location synchronization with safety conditions on the weight was considered, though without the requirement on the weight synchronization. Finally, synchronization under partial observability was recently studied in [13] but only in the context of finite automata without weights.

2 Definitions

We shall now formally define the synchronization problem on deterministic and complete weighted automata.

▶ **Definition 1** (Weighted Automaton). A (deterministic) *weighted automaton (WA)* is a tuple $\mathcal{A} = (L, Act, T, W)$ where

- L is a finite set of locations,
- Act is a finite set of actions,
- $T : L \times Act \to L$ is a transition function, and
- $W : L \times Act \to \mathbb{Z}$ is a weight function.

A *state* of $\mathcal{A}$ is a pair $(\ell, z) \in L \times \mathbb{Z}$ where ℓ is the current location and z the current weight. Let $S(\mathcal{A})$ be the set of all states of $\mathcal{A}$. We write $(\ell, z) \xrightarrow{a,w} (\ell', z')$ if $T(\ell, a) = \ell'$, $W(\ell, a) = w$ and $z' = z + w$.

A *path* in $\mathcal{A}$ is a finite sequence of states $\pi = s_0 s_1 \dots s_n$ such that for all i, $0 \leq i < n$, we have $s_i \xrightarrow{a_i,w_i} s_{i+1}$ for some $a_i \in Act$ and $w_i \in \mathbb{Z}$. The last state s_n in the path π is referred to as $last(\pi)$. The set of all paths is denoted by $Paths(\mathcal{A})$. For the complexity analysis in the rest of this paper, we assume a binary encoding of integers in $\mathcal{A}$.

▶ **Definition 2** (Observation Function). An *observation function* $\gamma : S(\mathcal{A}) \to \mathcal{O}$ maps each state of $\mathcal{A}$ to an observation from an observations set $\mathcal{O}$.

Assume now a given observation function γ to the set of observations $\mathcal{O}$. Let $\pi = s_0 s_1 \dots s_n$ be a path in $\mathcal{A}$. The observation function γ is naturally extended to an *observation sequence* for π by

$$\gamma(\pi) = \gamma(s_1)\gamma(s_2)\dots\gamma(s_n) \ .$$

▶ **Definition 3** (Strategy). A *strategy* is a function $\delta : \mathcal{O}^+ \to Act \cup \{\mathsf{done}\}$ that maps a nonempty sequence of observations to a proposed action or the symbol $\mathsf{done} \notin Act$, signaling that no further actions will be proposed.

A path $\pi = s_0 s_1 \dots s_n$ *follows* a strategy δ if $s_i \xrightarrow{a_i,w_i} s_{i+1}$ for $a_i = \delta(\gamma(s_0 s_1 \dots s_i))$ and $w_i \in \mathbb{Z}$, for all i, $0 \leq i < n$. A strategy δ is *terminating* if it does not generate any infinite path, in other words there is no infinite sequence where all its finite prefixes follow δ.

Given a subset of states $X \subseteq S(\mathcal{A})$ and a terminating strategy δ, the set of all maximal paths that follow the strategy δ in $\mathcal{A}$ and start from some state in X, denoted by $\delta[X]$, is defined as follows:

$$\delta[X] = \{\pi = s_0 s_1 \dots s_n \in Paths(\mathcal{A}) \mid s_0 \in X, \pi \text{ follows } \delta \text{ and } \delta(\gamma(\pi)) = \mathsf{done}\} \ .$$

The set of final states reached when following δ starting from X is defined as $last(\delta[X]) = \{last(\pi) \mid \pi \in \delta[X]\}$. Assuming a given observation function γ, we can now define a synchronizing strategy that will bring the system from any unknown initial state to the same single synchronizing state.

▶ **Definition 4** (Synchronization)**.** Given a WA $\mathcal{A}$, a strategy δ is *synchronizing* if δ is terminating and $|last(\delta[S(\mathcal{A})])| = 1$. Further, $\mathcal{A}$ is *synchronizable* if it admits a synchronizing strategy.

We limit our study to systems where we see no information about the current location and have a partial observability of the current weight so that we can distinguish whether its value is negative or nonnegative. This is the minimal possible observation as if we cannot observe anything about the weight then synchronization is impossible. Hence, we define the observation function γ to the set of observations $\mathcal{O} = \{<0, \geq 0\}$ by

$$\gamma((\ell, z)) = \begin{cases} <0 & \text{if } z < 0 \\ \geq 0 & \text{if } z \geq 0 \ . \end{cases}$$

We are interested in deciding whether a given WA is synchronizable under this observation function γ.

3 Polynomial Time Algorithm for Synchronizing

Let $\mathcal{A} = (L, Act, T, W)$ be a WA. We write $\ell \xrightarrow{a,w} \ell'$ for $\ell, \ell' \in L$ whenever $T(\ell, a) = \ell'$ and $w = W(\ell, a)$. A *cycle* in $\mathcal{A}$ starting in ℓ_0 is a path of the form $\ell_0 \xrightarrow{a_0,w_0} \ell_1 \xrightarrow{a_1,w_1} \dots \ell_n \xrightarrow{a_n,w_n} \ell_0$. The *weight* of the cycle is $\sum_{i=0}^{n} w_i$.

We now perform a series of checks in order to test whether we can synchronize from any possible initial state. The tests will give a necessary and sufficient condition for synchronization. At the end, we will argue that all the checks can be done in polynomial time. Therefore, we provide a polynomial time algorithm for deciding synchronizability. Furthermore, if all the checks succeed, we also construct a synchronizing strategy. It works in several phases, each concerning some of the tests. However, we note that although our algorithm decides synchronizability in polynomial time, the construction of a synchronizing strategy as an explicit function is not possible due to the fact that the lengths of the synchronizing sequences proposed by the strategy are unbounded (they depend on the initial weight values). We instead provide an algorithm, describing the unbounded strategy from any given initial state.

First, we check if the given $\mathcal{A}$, viewed as a labelled directed graph, has the following property.

Property 1. The graph $\mathcal{A}$ has a strongly connected component that is reachable from any location in $\mathcal{A}$.

If Property 1 is not satisfied, and such a bottom strongly connected component does not exist, clearly there is no synchronizing strategy for $\mathcal{A}$. From now on, assume that Property 1 holds. Taking advantage of this property, we define the first phase of the constructed synchronizing strategy δ.

▷ **Phase 1.** For every location ℓ let $home(\ell)$ be a sequence of actions that will bring ℓ into this strongly connected component and for any sequence of actions x let $\ell[x]$ be the location that we will reach from ℓ after performing the sequence x (note that this is well defined due to the fact that $\mathcal{A}$ is deterministic). Our synchronizing strategy will start by performing the action sequence $x_1x_2x_3 \dots x_n$ where

$$x_1 = home(\ell_1), x_2 = home(\ell_2[x_1]), x_3 = home(\ell_3[x_1x_2]), \dots, x_n = home(\ell_n[x_1x_2 \dots x_{n-1}])$$

assuming that $L = \{\ell_1, \ell_2, \ldots, \ell_n\}$. We shall refer to this technique as *sequentialization*: intuitively, even if the initial location is unknown, we can perform given actions for each possible initial location, meanwhile tracking where we move if the actual initial location was different, and execute these steps in a sequence for each possible location. ◁

▶ **Example 1.** We illustrate Phase 1 on the system below in Figure 2. We first execute $home(\ell_1) = a$. Meanwhile both ℓ_2 and ℓ_3 move to ℓ_2. Therefore, we proceed with $home(\ell_2) = ba$. Therefore, if we started in ℓ_3 we are now in ℓ_4, too. Since ℓ_4 is in a bottom strongly connected component, $home(\ell_4)$ is the empty sequence and we are done.

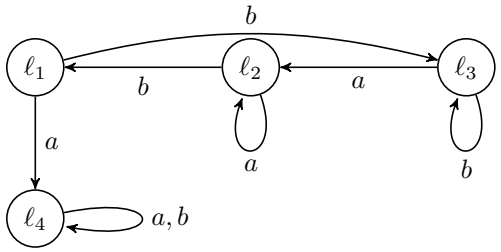

Figure 2 Example of sequentialization (the word aba will bring all locations to ℓ_4)

Consequently, after Phase 1, we are for sure in the strongly connected component. Within this component, we check the following property.

Property 2. Let $\mathcal{A}$ be strongly connected. In $\mathcal{A}$, there is a cycle with weight 1 and a cycle with weight -1.

▶ **Lemma 5.** *If Property 2 is not satisfied then there is no synchronizing strategy.*

Proof. Assume that $\mathcal{A}$ is synchronizable. Then there must be a positive and a negative cycle in $\mathcal{A}$. Further, for any location ℓ, the states $(\ell, 0)$ and $(\ell, 1)$ can be synchronized. Therefore, there is a path from both these states to some state (ℓ', z) for some $z \in \mathbb{Z}$. Since $\mathcal{A}$ is strongly connected, there is also a path from (ℓ', z) back to the state (ℓ, z') for some $z' \in \mathbb{Z}$. Consequently, there are two cycles from ℓ with weights z' and $z' - 1$, respectively; moreover, these weights can be chosen non-zero due to existence of a positive cycle. Hence the weights of these two cycles are relative primes and in combination with the presence of a positive and negative cycle in $\mathcal{A}$, this implies the existence of cycles with the weights $+1$ and -1. ◀

From now on, assume that $\mathcal{A}$ is strongly connected and Property 2 holds. Observe that, consequently, there are $+1$ and -1 cycles starting and ending in each location ℓ. Let us denote the corresponding sequences of actions by ℓ^+ and ℓ^-. The first, and rather naive, use of these cycles is to get the weight component of the state close to zero.

▷ **Phase 2.** We extend our strategy δ by performing the ± 1 cycles until we see a change in our observation. Assuming we start with nonnegative observation, Phase 2 ends at the moment when a negative observation is reached (and symmetrically for the other case). To this end, assuming $L = \{\ell_1, \ldots, \ell_n\}$, we employ the sequentialization technique again. We first execute the word ℓ_1^- for the -1 cycle from ℓ_1 and keep track of the resulting locations $\{\ell'_1, \ldots, \ell'_n\}$. Note that their weights could have increased instead, say by at most c_1. Next we execute ${\ell'_2}^-$ exactly $(c_1 + 1)$-times, so that even if the initial location was ℓ_2, after this many cycles the weight decreased no matter how it increased by performing ℓ_1^-. Meanwhile

ℓ_3' changes to ℓ_3'' and its weight could have in total increased by at most c_2. We thus execute $\ell_3''^-$ exactly (c_2+1)-times and so on for all locations cyclically (starting again at the first location once we went through all of them) until the weight decreases below zero. This process terminates since whenever performing cycles for a particular location, its weight (if we indeed started in the respective location) drops below any previous value. ◁

▶ **Example 2.** We illustrate Phase 2 on the system below in Figure 3 when the observation is nonnegative. We first execute $\ell^- = a$. Meanwhile r loops under a and increases by $c_1 = 3$. Therefore, we proceed with repeating $r^- = bab$ for 4 times. This in turn makes ℓ return again to ℓ with value increased by $c_2 = 4 \cdot 3 = 12$. Next we repeat ℓ^- for 13 times etc.

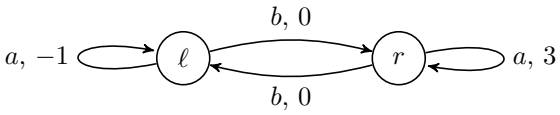

Figure 3 Example illustrating Phase 2

We are now guaranteed that right after Phase 2, the observation has just changed. Therefore, we are now in a state (ℓ, z) for some $\ell \in L$ and

$$0 \le z < M \text{ if } \gamma((\ell, z)) = \ge 0 \quad \text{or} \quad -M \le z < 0 \text{ if } \gamma((\ell, z)) = <0$$

where M is the largest absolute value of any weight used in $\mathcal{A}$. Therefore, there are finitely many states we can be at. Now that we have a bound how far we are from zero, we can make a better use of ± 1 cycles and derive for each location, where we possibly might be at, its weight.

▷ **Phase 3.** Once again, we employ sequentialization. Assuming first that we are in location $\ell_1 \in L$ of a state (ℓ_1, z) with $-M \le z < M$, we can perform a sequence of actions corresponding to -1 or $+1$ cycle from ℓ_1 (depending on whether $\gamma((\ell_1, z)) = \ge 0$ or $\gamma((\ell_1, z)) = <0$) until the observation changes at the end of the cycle when we are again in ℓ_1. If we indeed started in the location ℓ_1, we know that we are now in the state $(\ell_1, -1)$ if $\gamma((\ell_1, z)) = \ge 0$, or $(\ell_1, 0)$ if $\gamma((\ell_1, z)) = <0$. If the weight in the reached states did not change from nonnegative to negative (or the other way round) even after performing M cycles, we know for sure that we were not in the location ℓ_1. The situation where we started in a different location than ℓ_1 and the weight observation still changed as expected simply adds an extra (false) hypothesis that can be eliminated (as shown later on in this section).

Now consider what would happen, until now, if we were instead in location $\ell_2 \in L$ at the moment before we started to perform the -1 or $+1$ cycles for ℓ_1. After playing according to the strategy above, we would be now in a possibly different location ℓ_2' with weight in the range $[-M', M']$ where M' can be computed from M and the strategy performed so far. We can now start performing -1 or $+1$ cycle from ℓ_2' exactly as before in order to determine the exact weight in this location (provided we started in ℓ_2) and we continue like this with handling ℓ_3 etc., for each location in L. ◁

▶ **Example 3.** We illustrate Phase 3 on the system below in Figure 4. If the observation is nonnegative, the current weight is at most 2 and we start with repeating $bbaa$, a -1 cycle for ℓ, for at most $M = 3$ times. If the observation remains nonnegative after this sequence (case 1) we must be in r. Otherwise (case 2), we stop when the observation changes and

if we are in ℓ the current weight is -1. Meanwhile r returned back to r and could have increased its weight to at most 5 in case 2. Then we proceed with repeating *bbaa* at most 6 times to get for sure to $(r, -1)$. In case 1, the observation is negative and the weight is at least -2. Hence, we repeat *aab*, a $+1$ cycle for r. Say that the observation changes after two repetitions. Then we are either in $(r, 0)$ or in the meanwhile achieved $(\ell, -3)$. (The latter is, however, impossible here since the observation would remain negative.)

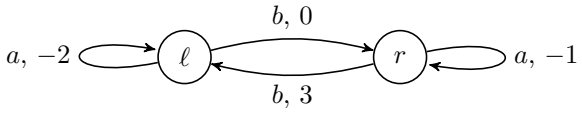

Figure 4 Example illustrating Phase 3

We conclude that after Phase 3 we must be in one of the states from the set

$$\{(\ell_1, z_1), (\ell_2, z_2), \ldots, (\ell_n, z_n)\}$$

called the *hypothesis set*, where all z_i's are exactly known. We can w.l.o.g. assume that all locations in the assumption are pairwise different. Indeed, we can perform a number of ± 1 cycles from the location that appears in the hypothesis set more times and determine which one of the weights is still feasible (at most one is). Note that the size of the hypothesis set is thus at most $|L|$.

The next task is to distinguish between these hypotheses. For each pair of locations, assuming their weights from the hypothesis set, there must be a way to synchronize them. We present three tests such that at least one of them must be passed by each pair. All tests refer to the following notion of difference graph.

▶ **Definition 6** (Difference Graph)**.** The *difference graph* of a WA $\mathcal{A}$ is a weighted graph $G^{\mathcal{A}} = (V, E)$ with $E \subseteq V \times \mathbb{Z} \times V$ such that $V = L \times L$, and for every $a \in Act$ we have $\big((\ell, \ell'), W(\ell, a) - W(\ell', a), (T(\ell, a), T(\ell', a))\big) \in E$.

In other words, $G^{\mathcal{A}}$ is a synchronous product of two $\mathcal{A}$'s, where each edge weight is the difference of edge weights in the first and the second component.

▶ **Example 4.** Consider the system on the upper part of Figure 5, parametrized by $k \in \mathbb{Z}$. We depict a part of its difference graph on the lower part of the figure.

We have already seen how to distinguish states with the same location using ± 1 cycles. For all pairs of locations (ℓ_1, ℓ_2) where $\ell_1 \neq \ell_2$, we run the following three tests.

Property 3. There is a path in $G^{\mathcal{A}}$ from (ℓ_1, ℓ_2) to (ℓ, ℓ) for some $\ell \in L$.

Property 4. There is a path in $G^{\mathcal{A}}$ from (ℓ_1, ℓ_2) to (ℓ_1', ℓ_2') such that there is a cycle of nonzero weight (positive or negative) in $G^{\mathcal{A}}$ starting in (ℓ_1', ℓ_2').

In order to define the last Property 5, we need additional notions and reasoning. If Property 4 is not satisfied for a given pair (ℓ_1, ℓ_2) then every cycle in $G^{\mathcal{A}}$ reachable from (ℓ_1, ℓ_2) has zero weight. Therefore, whenever any cycle C in $G^{\mathcal{A}}$ is performed in $\mathcal{A}$ starting from location ℓ_1 or ℓ_2, the weight changes in both cases by the same value, called the *projected weight* $\mathcal{P}(C)$ of C. Although $G^{\mathcal{A}}$ may be disconnected, we may w.l.o.g. assume that (ℓ_1, ℓ_2) is a node in $G^{\mathcal{A}}$ that is a part of some strongly connected component (otherwise, we bring it there, using sequentialization).

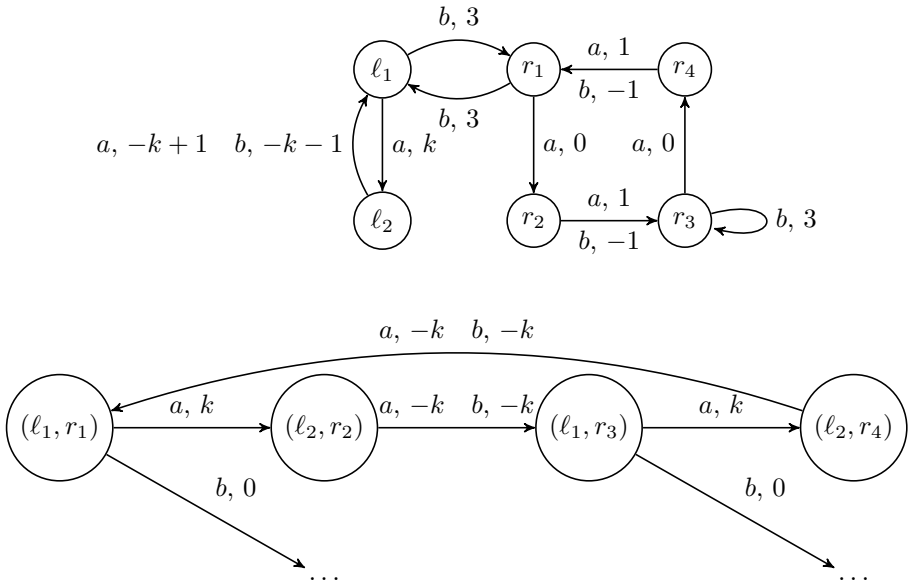

Figure 5 Example of difference graph

▶ **Lemma 7.** *For every strongly connected component S of $G^{\mathcal{A}}$ satisfying Property 2 and not satisfying Property 4, there is a number p such that $1 \leq p \leq |L|$ and for any node (pair of locations) in S there are cycles C^+, C^- starting in this node with $\mathcal{P}(C^+) = p$ and $\mathcal{P}(C^-) = -p$. Moreover, such p can be computed in polynomial time.*

Proof. Let (ℓ_1, ℓ_2) be an arbitrary node in S. Due to Property 2, there is a cycle from ℓ_1 in $\mathcal{A}$ with weight $+1$. We repeatedly perform this $+1$ cycle and follow the same behaviour from (ℓ_1, ℓ_2) in $G^{\mathcal{A}}$. At the end of each cycle, the first component in $G^{\mathcal{A}}$ will be in the location ℓ_1 and the second component in one of the $|L|$ possible locations. By the pigeon-hole principle, after performing the $+1$ cycle at most $|L|$ times, we will find a repeated pair in $G^{\mathcal{A}}$. Hence we found a cycle C^+ in $G^{\mathcal{A}}$ with zero weight in $G^{\mathcal{A}}$, due to the violation of Property 4, and with the projected weight $0 < \mathcal{P}(C^+) \leq |L|$. By the same arguments, but using the fact about the existence of -1 cycle in $\mathcal{A}$, we can find a cycle C^- in $G^{\mathcal{A}}$ with the projected weight $0 > \mathcal{P}(C^-) \geq -|L|$.

Let us now argue that for each node we can choose cycles with absolute weights equal to a fixed integer. Let $p_{\min} := \min\{|\mathcal{P}(C)| \mid C \text{ is a cycle in } S\}$ denote the smallest projection over all cycles in the strongly connected component S. We claim that for any pair of states (ℓ_1, ℓ_2) in S, there are cycles in S starting in (ℓ_1, ℓ_2) with projected weights $p_{\min}$ and $-p_{\min}$. Indeed, note that there is a cycle C in S with $|\mathcal{P}(C)| = p_{\min}$ and there are cycles D^+ and D^- from (ℓ, ℓ') that visit some state of C and have positive and negative projected weight, respectively. Now by repeating C on the way either in D^+ or in D^-, we construct cycles E^+ and E^- with $0 < \mathcal{P}(E^+) \leq p_{\min}$ and $0 > \mathcal{P}(E^-) \geq -p_{\min}$, respectively. By minimality of $p_{\min}$, we obtain $\mathcal{P}(E^+) = p_{\min}$ and $\mathcal{P}(E^-) = -p_{\min}$. Note that, moreover, for each cycle C is S, $\mathcal{P}(C)$ is a multiple of $p_{\min}$. And vice versa, for each multiple of $p_{\min}$, there is a cycle with such projected weight in any node of S. Therefore, we can perform the pigeon-hole construction of a cycle (in polynomial time), obtaining a weight $0 < p \leq |L|$, and we are guaranteed that from each node there are cycles with projected weights p and $-p$, respectively (although p is not necessarily minimal). ◀

Consequently, for each strongly connected component S, we have $0 < p \leq |L|$. For S, we define a reachability problem on a graph $\mathcal{A}_S = (V, \to)$ where

- $V = (L \times \{0, \ldots, p-1\}) \times (L \times \{0, \ldots, p-1\}) \cup \{\mathit{separated}\}$, and
- for each $v = \big((\ell_1, z_1), (\ell_2, z_2)\big)$ and $a \in \mathit{Act}$, let $v' = \big((\ell_1', z_1'), (\ell_2', z_2')\big)$ where $\ell_1' = T(\ell_1, a)$, $\ell_2' = T(\ell_2, a)$ and $z_1' = z_1 + W(\ell_1, a) + \alpha \cdot p$ and $z_2' = z_2 + W(\ell_2, a) + \alpha \cdot p$ for the unique $\alpha \in \mathbb{Z}$ such that the larger of z_1', z_2' lies in the interval $[0, p-1]$; we set
  - $v \to v'$ if $v' \in V$, i.e. the lower weight is also nonnegative, and
  - $v \to \mathit{separated}$, otherwise, i.e. the lower weight is negative.

We say that the graph $\mathcal{A}_S$ is *distinguishing* for a pair of locations $(\ell_1, \ell_2) \in S$ if from any initial node $\big((\ell_1, z_1), (\ell_2, z_2)\big)$, for each $0 \leq z_1, z_2 \leq p-1$, we can reach the node *separated*. Note that the size of $\mathcal{A}_S$ is at most $|L|^4 + 1$, hence polynomial in $|\mathcal{A}|$. Now we state the final test.

Property 5. If (ℓ_1, ℓ_2) belongs to a strongly connected component S of $G^{\mathcal{A}}$ then the graph $\mathcal{A}_S$ is distinguishing for (ℓ_1, ℓ_2).

▶ **Example 5.** Consider the difference graph of Figure 5. Observe that there is no path from (ℓ_1, r_1) to a pair with identical components, as well as no nonzero cycle. The length p is equal 2 here. If $k \geq 2$ then we have $((\ell_1, 0), (r_1, 0)) \to \mathit{separated}$ as action a immediately creates a large enough difference. If $k = 1$, then *separated* is still reachable from $((\ell_1, 0), (r_1, 0))$, but only after aa is taken. Then both weights are 1 and the next action a creates the distinguishing difference as the weights would now be $2, 1$, i.e. transformed to $0, -1$.

Supposing each pair of locations satisfies Property 3 or Property 4 or Property 5, we can iteratively decrease the size of the hypothesis set until it becomes a singleton as shown in the next phase.

▷ **Phase 4.** We employ sequentialization again. We pick any two states from the current hypothesis set and eliminate at least one of them as described below. Meanwhile, we update all remaining states from the hypothesis set to their current states. We repeat this procedure until the hypothesis set becomes a singleton. Let (ℓ_1, z_1) and (ℓ_2, z_2) be the currently explored pair from the hypothesis set.

First, if Property 3 holds we perform the sequence of actions that brings both locations into a single location. Afterwards, if their respective weights are different, using the ± 1 cycles, we detect at least one of the weights impossible as above. Thus we decrease the size of the hypothesis set.

Second, if Property 4 holds then we can extend our strategy δ by executing the sequence of actions that brings (ℓ_1, ℓ_2) to some (ℓ_1', ℓ_2') where we can repeatedly execute actions on the nonzero cycle in $G^{\mathcal{A}}$ until the weights in the pair of states reached after this sequence are sufficiently (see below) far away from each other. Assume w.l.o.g. that the weights z_1', z_2' of the two reached states are both positive and $z_1' < z_2'$ (the other situations are symmetric). Now from the location with the lower weight (ℓ_1'), we enter a simple cycle in $\mathcal{A}$ with the minimal (negative) weight and start executing it. This ensures that if we started from ℓ_1 or ℓ_2, then the observation will change to negative in n_1 or n_2 steps, respectively, where $n_1 < n_2$ (since $|z_2' - z_1'|$ was sufficiently large) and we can compute these numbers. If the observation changes after exactly n_1 steps, we eliminate the state corresponding to ℓ_2 from the hypothesis set. If the observation changes after exactly n_2 steps, we eliminate the state corresponding to ℓ_1 from the hypothesis set. If the observation changes after a different number of steps, we eliminate both.

Third, let Property 5 hold (and Property 4 not) and (ℓ_1, ℓ_2) be in a strongly connected component S of $G^{\mathcal{A}}$. By Lemma 7, we have zero cycles in $\mathcal{A}_S$ with projected weights p and $-p$ where $0 < p \leq |L|$. We perform these cycles until the larger weight is in $[0, p-1]$. If the lower weight is negative at this moment, the current observation eliminates one of the hypotheses. Otherwise, the weights in both states are in $[0, p-1]$. Due to Property 5 we have a strategy to reach *separated* in $\mathcal{A}_S$, inducing a strategy in $\mathcal{A}$ by inserting the $-p$ and p cycles. Upon reaching *separated* in $\mathcal{A}_S$, the observation in $\mathcal{A}$ proves one of the two hypotheses impossible. (If at any moment throughout the process, an unexpected change of observation occurs, we eliminate the respective hypothesis from the set immediately.)

Once the hypothesis set is a singleton, we know precisely the current state. Finally, we deterministically reach a fixed location and fixed weight (by performing ± 1 cycles) and thus synchronize. ◁

The stated properties are not only sufficient, but also necessary conditions for synchronizability:

▶ **Lemma 8.** *Let $\mathcal{A}$ be a strongly connected WA satisfying Property 2. Then $\mathcal{A}$ is synchronizable if and only if for each pair of locations (ℓ_1, ℓ_2) either Property 3 or Property 4 or Property 5 is satisfied.*

Proof. The "if"-part follows from the previously constructed synchronizing strategy. For the "only-if"-part, assume that there is a pair (ℓ_1, ℓ_2) satisfying neither Property 3, nor Property 4, nor Property 5. By the last one, there are weights $z_1, z_2 \in [0, p-1]$ such that the node *separated* is not reachable from the configuration $\mathit{init} = \big((\ell_1, z_1), (\ell_2, z_2)\big)$ in the graph $\mathcal{A}_p$. For a contradiction, assume that $\mathcal{A}$ admits a synchronizing strategy σ. When σ is applied to initial states (ℓ_1, z_1) and (ℓ_2, z_2), we obtain two paths π_1 and π_2, inducing two sequences of observations $\gamma(\pi_1)$ and $\gamma(\pi_2)$. Comparing the respective elements in the two sequences, there are two cases.

In the first case, observations will never differ. Since σ is synchronizing, it brings both states (ℓ_1, z_1) and (ℓ_2, z_2) eventually into the same state, in particular to the same location, witnessing Property 3 and contradicting to our assumption.

In the second case, after a certain number of steps, the observations of the current states (ℓ_1', z_1') and (ℓ_2', z_2') of the two path will differ, w.l.o.g. $z_1' < 0 \leq z_2'$. Since Property 4 is not satisfied, by Lemma 7 there are cycles increasing and decreasing weight in both ℓ_1 and ℓ_2 by p. The two paths π_1, π_2 produced by the strategy σ in $\mathcal{A}$ induce two sequences $\hat{\pi}_1, \hat{\pi}_2$ where the ith elements are both increased/decreased by $\alpha_i \cdot p$ for some $\alpha_i \in \mathbb{Z}$ so that the larger one is in $[0, p-1]$. These sequences straightforwardly induce a path in $\mathcal{A}_S$, where S is the strongly connected component of *init*. Since *separated* cannot be reached from *init*, the smaller weight is always in $[0, p-1]$, too. Let $\hat{z}_1', \hat{z}_2'$ denote the weights in $\mathcal{A}_S$ when σ achieves z_1', z_2'. Since $z_2' \geq 0$, $\hat{z}_2' < p$, and $\hat{z}_2' \equiv z_2' \pmod{p}$, we obtain $\hat{z}_2' \leq z_2'$. Therefore, by $\hat{z}_2' - \hat{z}_1' = z_2' - z_1'$ we also get $\hat{z}_1' \leq z_1'$. Since $z_1' < 0$, we obtain $\hat{z}_1' < 0$, a contradiction. ◀

4 Complexity

We can now state our main theorem:

▶ **Theorem 9.** *The synchronizability problem for deterministic weighted automata is decidable in polynomial time.*

Proof. Properties 1-5 form sufficient and necessary conditions for the existence of a synchronizing strategy for $\mathcal{A}$ by Lemma 5 and Lemma 8. Moreover, all properties can be verified in

polynomial time. Indeed, the size of $G^{\mathcal{A}}$ is polynomial in $|\mathcal{A}|$ and p necessary for constructing $\mathcal{A}_S$ is computable in polynomial time by Lemma 5, and the presence of ± 1 cycles is decided in polynomial time by Theorem 11 as discussed in the rest of this section. ◀

We now prove that the presence of ± 1 cycles can be decided in polynomial time. We assume a weighted graph $G = (V, E)$ where V is a finite set of nodes and $E \subseteq V \times \mathbb{Z} \times V$ are the edges written as $u \xrightarrow{w} v$ whenever $(u, w, v) \in E$. A *path* in G is a sequence of edges $v_0 \xrightarrow{w_0} v_1 \xrightarrow{w_1} \dots \xrightarrow{w_{n-1}} v_n$. A *weight of a path* π is defined as $|\pi| = \sum_{i=0}^{n-1} w_i$. A *$k$-cycle* is a path π where $v_0 = v_n$ such that $k = |\pi|$.

▶ Remark. We first briefly discuss related problems and point to severe differences, preventing us from adapting the existing results. On the one hand, we note that the problem whether there is a k-cycle, where k is a part of the input, is NP-hard (see full version of the paper). On the other hand, it is a classical result [12] that existence of 0-cycles is decidable in polynomial time. The result can be proven by a reduction to linear programming. The idea is the following. For each transition, there is a variable encoding the frequency of the transition on the desired cycle. Encoding of Kirchhoff's flow-preservation laws then ensures that the frequencies indeed induce a cycle. Finally, the sum of transition weights multiplied by the frequencies is required to be 0. From every rational solution, we can by multiplication obtain an integer solution, and thus a realizable cycle. Since 0 multiplied by any number remains 0, we thus obtain a 0-cycle. In contrast, in our setting, this idea cannot be used. Indeed, suppose we require the frequency-weighted sum of edge-weights to be 1. Since the frequencies and thus also the number 1 must be multiplied by an a priori unknown integer, in order to obtain an integer solution, the resulting total weight is not 1. Asking instead directly for an integer solution to the system is an instance of integer linear programming, which is an NP-hard problem. Instead of using linear programming, we employ (as shown in the full version of the paper) a number theoretic arguments and exploit Dijkstra's shortest path algorithm on graphs where weights are counted modulo various numbers. Finally, note that although we can decide the existence of ± 1-cycles in polynomial time, the length (number of edges) of the shortest one may still be exponential. For instance, consider a single vertex with two self-loops labelled by $2^n + 1$ and -2. △

The discussion suggests that number theoretic techniques have to be applied. We reduce our problem to the problem whether the greatest common divisor of all cycles in a graph is 1. Formally, for a weighted graph G, let the *period* $\gcd(G)$ denote $\gcd\{k \mid k \in \mathbb{Z}, G \text{ has a } k\text{-cycle}\}$.

▶ **Proposition 10.** For every strongly connected weighted graph G, there is a 1-cycle and a -1-cycle in G if and only if there is a positive and a negative cycle in G and $\gcd(G) = 1$.

Proof. The 'Only-if' direction is trivial. For the 'If' direction, $\gcd(G) = 1$ yields by Bézout's identity an equality

$$1 = \alpha_1 \cdot k_1 + \dots + \alpha_n \cdot k_m \tag{1}$$

for some $m \in \mathbb{N}$, $\alpha_i \in \mathbb{Z}$, and k_i being the weight of some cycle c_i in G, and where, moreover, some $k_p > 0$ and some $k_n < 0$. Note that these numbers can be extracted using the extended Euclidean algorithm. First, we argue, we can choose all $\alpha_i \geq 0$ so that Equation (1) still holds.

Whenever $\alpha_i < 0$ with k_i positive, we increase α_i by $x \cdot (-k_n)$ for some $x \in \mathbb{N}$ so that it becomes positive. Further, we increase α_n by $x \cdot k_i$, thus preserving Equation (1). For

negative k_i, we proceed similarly, using k_p and α_p instead. Since this procedure only increases α's, they all eventually become positive.

Nonnegative coefficients α_i determine the number of repetitions of each cycle c_i. Since these cycles may be disconnected, this does not yield a single 1-cycle yet. To this end, we consider a negative cycle visiting each vertex of G, guaranteed by assumptions. Let $-\omega$ denote its weight. We construct a 1-cycle by executing this cycle and on the way, whenever reaching a vertex where the cycle c_i originates, we execute c_i for $(\omega+1)\cdot\alpha_i$ times. A -1-cycle is constructed similarly. ◀

▶ **Theorem 11.** *The presence of both a* 1*-cycle and at the same time a* -1*-cycle in a weighted graph* G *is decidable in polynomial time. Moreover, such cycles can be effectively constructed.*

Proof. Deciding presence of a negative cycle and producing a witness can be done in polynomial time using, for instance, Bellman-Ford algorithm (see e.g. [3]); the same holds for positive cycles by swapping the signs.

The period of a graph can be computed in polynomial time, too. Indeed, the result for unweighted graphs (all weights are one) was proven in [11]. Further, [1] suggests an extension of the technique to weighted graphs. Since [11] is to the best of our knowledge not accessible electronically (the only hardcopy of the report is located at library of Stanford University) and the correctness of the extension to weighted graphs is not proven in [1], we also provide our own proof, using supposedly different techniques. Full version of the paper gives the details. ◀

▶ Remark. The polynomial time algorithm for deciding synchronizability is relying only a single observation, testing whether the accumulated weight is negative or nonnegative. In a more general setting, we may consider a richer set of observations checking whether the weights are less-than/greater-or-equal to a given number of integer values. The techniques in this paper can be directly reused to handle this more generation situation and the only check that must be modified is Property 5. Here, if some observations are far away from each other (the integers that they test have distance more than p) then it is sufficient to check if at least of them succeeds, otherwise the graph $\mathcal{A}_S$ is extended to include weights in the range $[0, kp-1]$ where k is the number of observations that are close to each other so that all of them are considered in the check for distinguishability of a given pair of locations. As the observations are part of the input (of the problem description), this still creates a graph with only polynomially many nodes.

5 Conclusion

We have shown that the synchronization problem for deterministic WA under (minimal) partial observability is decidable in polynomial time. This result is based on a polynomial time algorithm for deciding the existence of $+1$ and -1 cycles in a weighted graph and states five necessary and sufficient conditions for synchronizability. All conditions are verifiable in polynomial time, despite the fact that the length of the resulting synchronization strategy is unbounded (as it depends on the initial weight values). The presented techniques are general and allow for a straightforward adaptation to the situation when more observations become available. Future research will include nontrivial extensions to nondeterministic WA and synchronization under safety constraints, e.g. constraints on the weight-levels encountered during the synchronization.

Acknowledgments.

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement 601148 (CASSTING), EU FP7 FET project SENSATION, Sino-Danish Basic Research Center IDAE4CPS, the European Research Council (ERC) under grant agreement 267989 (QUAREM), the Austrian Science Fund (FWF) project S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award), the Czech Science Foundation under grant agreement P202/12/G061, and People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) REA Grant No 291734.

References

1 Esther M. Arkin, Christos H. Papadimitriou, and Mihalis Yannakakis. Modularity of cycles and paths in graphs. *J. ACM*, 38(2):255–274, 1991.

2 Paolo Baldan and Daniele Gorla, editors. *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*. Springer, 2014.

3 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2nd edition, 2008.

4 Ján Černý. Poznámka k. homogénnym experimentom s konecnými automatmi. *Mat. fyz. čas SAV*, 14:208–215, 1964.

5 Laurent Doyen, Line Juhl, Kim Guldstrand Larsen, Nicolas Markey, and Mahsa Shirmohammadi. Synchronizing words for weighted and timed automata. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 121–132. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

6 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Infinite synchronizing words for probabilistic automata. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 278–289. Springer, 2011.

7 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Synchronizing objectives for markov decision processes. In Johannes Reich and Bernd Finkbeiner, editors, *iWIGP*, volume 50 of *EPTCS*, pages 61–75, 2011.

8 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Robust synchronization in markov decision processes. In Baldan and Gorla [2], pages 234–248.

9 Fedor Fominykh and Mikhail Volkov. P(l)aying for synchronization. In *Implementation and Application of Automata*, volume 7381 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2012.

10 Ramesh Hariharan, Telikepalli Kavitha, and Kurt Mehlhorn. Faster algorithms for minimum cycle basis in directed graphs. *SIAM J. Comput.*, 38(4):1430–1447, 2008.

11 Donald Knuth. Strong components. Technical Report 004639, Comput. Sci. Dept., Stanford University, Stanford, Calif., 1973.

12 S. Rao Kosaraju and Gregory F. Sullivan. Detecting cycles in dynamic graphs in polynomial time (preliminary version). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 398–406. ACM, 1988.

13 Kim Guldstrand Larsen, Simon Laursen, and Jiří Srba. Synchronizing strategies under partial observability. In Baldan and Gorla [2], pages 188–202.

14 Sven Sandberg. Homing and synchronizing sequences. In *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, pages 5–33. Springer, 2004.
15 Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, 2006.
16 Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In *Language and automata theory and applications*, pages 11–27. Springer, 2008.

Appendix

A Proofs

Lemma 5. *If Property 2 is not satisfied then there is no synchronizing strategy.*

Proof. Assume that $\mathcal{A}$ is synchronizable. Then there must be a positive and a negative cycle in $\mathcal{A}$. Further, for any location ℓ, the states $(\ell, 0)$ and $(\ell, 1)$ can be synchronized. Therefore, there is a path from both these states to some state (ℓ', z) for some $z \in \mathbb{Z}$. Since $\mathcal{A}$ is strongly connected, there is also a path from (ℓ', z) back to the state (ℓ, z') for some $z' \in \mathbb{Z}$. Consequently, there are two cycles from ℓ with weights z' and $z' - 1$, respectively; moreover, these weights can be chosen non-zero due to existence of a positive cycle. Hence the weights of these two cycles are relative primes and in combination with the presence of a positive and negative cycle in $\mathcal{A}$, this implies the existence of cycles with the weights $+1$ and -1. ◀

Lemma 7. *For every strongly connected component S of $G^{\mathcal{A}}$ satisfying Property 2 and not satisfying Property 4, there is p such that $1 \leq p \leq |L|$ and for any node (pair of locations) in S there are cycles C^+, C^- starting in this node with $\mathcal{P}(C^+) = p$ and $\mathcal{P}(C^-) = -p$. Moreover, such p can be computed in polynomial time.*

Proof. Let (ℓ_1, ℓ_2) be an arbitrary node in S. Due to Property 2, there is a cycle from ℓ_1 in $\mathcal{A}$ with weight $+1$. We repeatedly perform this $+1$ cycle and follow the same behaviour from (ℓ_1, ℓ_2) in $G^{\mathcal{A}}$. At the end of each cycle, the first component in $G^{\mathcal{A}}$ will be in the location ℓ_1 and the second component in one of the $|L|$ possible locations. By pigeon-hole principle, after performing the $+1$ cycle at most $|L|$ times, we will find a repeated pair in $G^{\mathcal{A}}$. Hence we found a cycle C^+ in $G^{\mathcal{A}}$ with zero weight in $G^{\mathcal{A}}$, due to the violation of Property 4, and with the projected weight $0 < \mathcal{P}(C^+) \leq |L|$. By the same arguments, but using the fact about the existence of -1 cycle in $\mathcal{A}$, we can find a cycle C^- in $G^{\mathcal{A}}$ with the projected weight $0 > \mathcal{P}(C^-) \geq -|L|$.

Let us now argue that for each node we can choose cycles with absolute weights equal to a fixed integer. Let $p_{\min} := \min\{|\mathcal{P}(C)| \mid C \text{ is a cycle in } S\}$ denote the smallest projection over all cycles in the strongly connected component S. We claim that for any pair of states (ℓ_1, ℓ_2) in S, there are cycles in S starting in (ℓ_1, ℓ_2) with projected weights $p_{\min}$ and $-p_{\min}$. Indeed, note that there is a cycle C in S with $|\mathcal{P}(C)| = p_{\min}$ and there are cycles D^+ and D^- from (ℓ, ℓ') that visit some state of C and have positive and negative projected weight, respectively. Now by repeating C on the way either in D^+ or in D^-, we construct cycles E^+ and E^- with $0 < \mathcal{P}(E^+) \leq p_{\min}$ and $0 > \mathcal{P}(E^-) \geq -p_{\min}$, respectively. By minimality of $p_{\min}$, we obtain $\mathcal{P}(E^+) = p_{\min}$ and $\mathcal{P}(E^-) = -p_{\min}$. Note that, moreover, for each cycle C is S, $\mathcal{P}(C)$ is a multiple of $p_{\min}$. And vice versa, for each multiple of $p_{\min}$, there is a cycle with such projected weight in any node of S. Therefore, we can perform the pigeon-hole construction of a cycle (in polynomial time), obtaining a weight $0 < p \leq |L|$, and we are guaranteed that from each node there are cycles with projected weights p and $-p$, respectively (although p is not necessarily minimal). ◀

Lemma 8. *Let $\mathcal{A}$ be a strongly connected WA satisfying Property 2. Then $\mathcal{A}$ is synchronizable if and only if for each pair of locations (ℓ_1, ℓ_2) either Property 3 or Property 4 or Property 5 is satisfied.*

Proof. The "if"-part follows from the previously constructed synchronizing strategy. For the "only-if"-part, assume that there is a pair (ℓ_1, ℓ_2) satisfying neither Property 3, nor Property 4, nor Property 5. By the last one, there are weights $z_1, z_2 \in [0, p-1]$ such that the node *separated* is not reachable from the configuration $\mathit{init} = \big((\ell_1, z_1), (\ell_2, z_2)\big)$ in the graph $\mathcal{A}_p$. For a contradiction, assume that $\mathcal{A}$ admits a synchronizing strategy σ. When σ is applied to initial states (ℓ_1, z_1) and (ℓ_2, z_2), we obtain two paths π_1 and π_2, inducing two sequences of observations $\gamma(\pi_1)$ and $\gamma(\pi_2)$. Comparing the respective elements in the two sequences, there are two cases.

In the first case, observations will never differ. Since σ is synchronizing, it brings both states (ℓ_1, z_1) and (ℓ_2, z_2) eventually into the same state, in particular to the same location, witnessing Property 3 and contradicting to our assumption.

In the second case, after a certain number of steps, the observations of the current states (ℓ_1', z_1') and (ℓ_2', z_2') of the two path will differ, w.l.o.g. $z_1' < 0 \leq z_2'$. Since Property 4 is not satisfied, by Lemma 7 there are cycles increasing and decreasing weight in both ℓ_1 and ℓ_2 by p. The two paths π_1, π_2 produced by the strategy σ in $\mathcal{A}$ induce two sequences $\hat{\pi}_1, \hat{\pi}_2$ where the ith elements are both increased/decreased by $\alpha_i \cdot p$ for some $\alpha_i \in \mathbb{Z}$ so that the larger one is in $[0, p-1]$. These sequences straightforwardly induce a path in $\mathcal{A}_S$, where S is the strongly connected component of *init*. Since *separated* cannot be reached from *init*, the smaller weight is always in $[0, p-1]$, too. Let $\hat{z}_1', \hat{z}_2'$ denote the weights in $\mathcal{A}_S$ when σ achieves z_1', z_2'. Since $z_2' \geq 0$, $\hat{z}_2' < p$, and $\hat{z}_2' \equiv z_2' \pmod p$, we obtain $\hat{z}_2' \leq z_2'$. Therefore, by $\hat{z}_2' - \hat{z}_1' = z_2' - z_1'$ we also get $\hat{z}_1' \leq z_1'$. Since $z_1' < 0$, we obtain $\hat{z}_1' < 0$, a contradiction. ◀

Lemma 17. *Every shortest non-Dijkstra path from u to v is*

a) either of the form $\pi_s \xrightarrow{w} v$, where π_s is a shortest path from u to s,
b) or $\rho_s \xrightarrow{w} v$ where ρ_s is a shortest non-Dijkstra path from u to s.

Proof. Suppose $\sigma \xrightarrow{w} v$ is a shortest non-Dijkstra path from u to v and the condition a) does not hold, i.e. $|\sigma| > |\pi_s|$ where $s = \mathit{last}(\sigma)$. We want to prove that then condition b) must hold. Since $\pi_s \xrightarrow{w} v$ is shorter than a shortest non-Dijkstra path $\sigma \xrightarrow{w} v$, it cannot be a non-Dijkstra path and hence $|\pi_s| + w \equiv |\pi_v| \pmod p$. Therefore, for any non-Dijkstra path σ' from u to s, i.e. with $|\sigma'| \not\equiv |\pi_s| \pmod p$, we thus obtain $|\sigma'| + w \not\equiv |\pi_v| \pmod p$, implying $\sigma' \xrightarrow{w} v$ is a non-Dijkstra path. Consequently, the non-Dijkstra path σ must be a shortest one. ◀

B Algorithm for Finding Period $\gcd(G)$ of Graph G

We show how to compute the period (gcd of all cycles) of a weighted graph. Although it is actually sufficient to examine simple cycles in the graph, there are still exponentially many of them. Hence, we compute $\gcd(G)$ in a more efficient way by Algorithm 1.

The idea is to pick any cycle, say with weight ω (for simplicity of notation, assume it is positive). If we knew the primal decomposition $\omega = p_1^{k_1} \cdots p_n^{k_n}$, we could search, for each p_i, for a cycle with weight not divisible by p_i. If we find such a cycle for each i, then $\gcd(G) = 1$. However, prime decomposition is not known to be computable in polynomial time. Therefore,

we look instead for a cycle with an arbitrary weight $\bar{\omega}$ not divisible by ω, i.e. $\bar{\omega} \not\equiv 0 \pmod{\omega}$. We use $\gcd(\omega, \bar{\omega})$ as the new ω and iterate this procedure until all cycles have weight divisible by the current ω. Intuitively, $\bar{\omega}$ eliminates at least one factor from the primal decomposition, although we cannot upfront determine which one.

Algorithm 1 Computation of $\gcd(G)$ for a weighted graph G

Input: Weighted graph $G = (V, E)$
Output: $\gcd(G)$
1: Pick an arbitrary simple cycle in G with positive weight, denoted by ω
2: **while** there is $\bar{\omega}$-cycle with $\bar{\omega} \not\equiv 0 \pmod{\omega}$ **do** ▷ $\gcd(\bar{\omega}, \omega) \neq \omega$
3: $\quad \omega \leftarrow \gcd(\omega, \bar{\omega})$ ▷ using e.g. Euclidean algorithm
4: **return** ω

The invariant of Algorithm 1 is that $\gcd(G)$ divides ω, following from $\gcd(G) = \gcd(\gcd(G), \gcd(\omega, \bar{\omega}))$ for any $\bar{\omega}$-cycle. Further, whenever $\omega \neq \gcd(G)$, there is an $\bar{\omega}$-cycle with $\gcd(\bar{\omega}, \omega) \neq \omega$, which is thus found in line 2, implying partial correctness of the algorithm. Termination follows from the unique finite primal decomposition by the fundamental theorem of arithmetic.

We now implement the test in line 2, returning the value $\bar{\omega}$, by Algorithm 2. Instead of finding an $\bar{\omega}$-cycle with $\bar{\omega} \not\equiv 0 \pmod{\omega}$, we decompose the cycle into an edge and the remaining path that form the cycle.

Algorithm 2 Detection of a cycle with weight not divisible by ω

Input: Weighted graph $G = (V, E)$, a number $\omega \in \mathbb{N}$
Output: Weight $\bar{\omega}$ of some cycle in G with $\bar{\omega} \not\equiv 0 \pmod{\omega}$, ff if there is none
1: **for all** $u \xrightarrow{x} v$ in G **do**
2: $\quad$ **if** there is a y-path from v to u with $y \not\equiv -x \pmod{\omega}$ **then**
3: $\quad\quad$ **return** $x + y$
4: **return** ff

This small trick allows us to use a modification of Dijkstra's algorithm for shortest paths. This modification is inspired by [10], where a similar idea was used in the context of minimum cycle bases.

▶ **Definition 16.** For $d \in \mathbb{Z}$ and $p \in \mathbb{N}$, a path π is a (d, p)-path if $|\pi| \not\equiv d \pmod{p}$.

In line 2 of Algorithm 2, we ask for the existence of a $(-x, \omega)$-path. We find more convenient to be more specific and, in Algorithm 3, we compute in some sense the shortest (d, p)-path. In order to avoid problems with negative weights in the context of shortest paths, we change every edge $u \xrightarrow{w} v$ in G into $u \xrightarrow{w'} v$ where $w' = w \bmod p$, i.e. we modify its weight so that the new weight w' belongs to the interval $[0, p)$ and satisfies $w' \equiv w \pmod{p}$. Clearly, the set of (d, p)-paths stays the same under this transformation. We denote the resulting weighted graph by $G^{\bmod p}$.

Given vertices *start* and *target*, Algorithm 3 first computes the shortest paths π_v from *start* to every node v in $G^{\bmod p}$. If the shortest path to *target* happens to be a (d, p)-path, we are done. Otherwise, for every vertex v, we compute the *shortest non-Dijkstra path*, i.e. the shortest path to v with a remainder modulo p different from that of $|\pi_v|$. Technically, in a Dijkstra-like computation, key_v stores the weight of the current shortest non-Dijkstra path to v, i.e. $key_v \not\equiv |\pi_v| \pmod{p}$. This ensures that we always have an alternative remainder to

Algorithm 3 Dijkstra's algorithm modified for the shortest (d, p)-path

Input: Weighted graph $G = (V, E)$, $start, target \in V$, $d \in \mathbb{Z}$, $p \in \mathbb{N}$
Output: Weight of a shortest path π in $G^{\bmod p}$ from $start$ to $target$ with $|\pi| \not\equiv d \pmod p$, ff if there is none

1: $G \leftarrow G^{\bmod p}$ ▷ constructing $G^{\bmod p}$ with only positive weights
2: for each $v \in V$, compute the shortest path π_v from $start$ to v ▷ using e.g. Dijkstra's algorithm
3: **if** $|\pi_{target}| \not\equiv d \pmod p$ **then**
4: **return** $|\pi_{target}|$
5: **else**
6: **for all** $v \in V$ **do**
7: $key_v \leftarrow \min\{|\pi_u| + w \mid u \xrightarrow{w} v,\ |\pi_u| + w \not\equiv |\pi_v| \pmod p\}$ ▷ $\min \emptyset = \infty$
8: enqueue all $v \in V$ with $key_v < \infty$ to the priority queue Q ordered by key
9: **while** $Q \neq \emptyset$ **do**
10: $u \leftarrow$ dequeue from Q
11: **for all** $u \xrightarrow{w} v$ **do**
12: $k \leftarrow key_u + w$
13: **if** $k < key_v$ and $k \not\equiv |\pi_v| \pmod p$ **then**
14: $key_v \leftarrow k$
15: enqueue v to Q
16: **if** $key_{target} = \infty$ **then return** ff
17: **else return** key_{target}

the one given by $|\pi_v|$. Consequently, either $|\pi_v|$ or key_v will always have a remainder modulo p different from d. Formally, correctness of Algorithm 3 is a consequence of the following lemma:

▶ **Lemma 17.** *Every shortest non-Dijkstra path from u to v is*

a) either of the form $\pi_s \xrightarrow{w} v$, where π_s is a shortest path from u to s,
b) or $\rho_s \xrightarrow{w} v$ where ρ_s is a shortest non-Dijkstra path from u to s.

In summary, Algorithm 2 decides in polynomial time for a given graph G and a weight ω whether G has a $\bar{\omega}$-cycle such that $\bar{\omega} \not\equiv 0 \pmod \omega$. The use of Algorithm 3 as a subroutine for line 2 is correct due to the fact that (i) every cycle in G has the same value in $G^{\bmod \omega}$ modulo ω and (ii) whenever there is a (d, p)-path between two nodes, there is also a shortest one.

C Detecting k-Cycles in Weighted Graphs is NP-Hard

We want to show that given a weighted graph G and an integer k, the existence of a k-cycle in G is NP-hard. The folklore reduction from the subset sum problem [15] to the existence of a path of a given weight k in a weighted graph must be slighly modified in order to work also for cycles.

▶ **Lemma 18.** *Given a weighted graph G and an integer k, the problem whether G contains a k-cycle is NP-hard.*

Proof. We provide a reduction from the subset sum problem. Given an instance of subset sum $(\{x_1, x_2, \ldots, x_n\}, T)$, where $T \in \mathbb{N}$ and $x_i \in \mathbb{N}$ for all i, $1 \leq i \leq n$, the subset sum

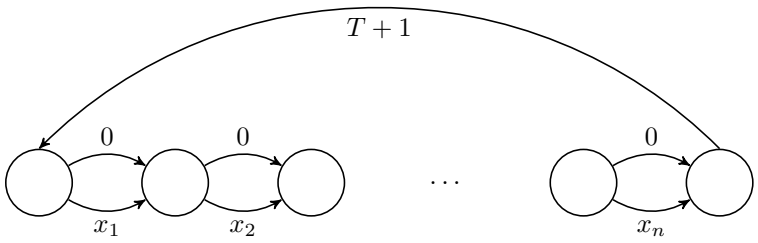

Figure 6 Subset sum is solvable if and only if there is a $(2T+1)$-cycle

question is whether there is a subset $X \subseteq \{x_1, x_2, \ldots, x_n\}$ such that $\sum X = T$. From the subset sum instance, we construct a weighted graph G as in Fig. 6. Clearly, the subset sum instance has a solution if and only if there is a $(2T+1)$-cycle in G. Notice that the extra edge with weight $T+1$ is necessary as it allows to take the cycle only once in order to compose the remaining number T from the edges with weights x_i, $1 \leq i \leq n$. Should the cycle be taken more than once, its weight will for sure be at least $2T+2$ and hence it will not be a $(2T+1)$-cycle. ◀