On the Complexity of Variants of the k Best Strings Problem

Martin Berglund and Frank Drewes

Department of Computing Science, Umeå University
90187 Umeå, Sweden
mbe@cs.umu.se, drewes@cs.umu.se

Abstract. We investigate the problem of extracting the k best strings from a non-deterministic weighted automaton over a semiring $\mathbb{S}$. This problem, which has been considered earlier in the literature, is more difficult than extracting the k best runs, since distinct runs may not correspond to distinct strings. Unsurprisingly, the computational complexity of the problem depends on the semiring $\mathbb{S}$ used. We study three different cases, namely the tropical and complex tropical semirings, and the semiring of positive real numbers. For the first case, we establish a polynomial algorithm. For the second and third cases, NP-completeness and undecidability results are shown.

1 Introduction

Weighted finite-state automata (WFA) are a popular tool for representing weights assigned to potentially infinite languages of strings. This is useful in many areas, notable cases being natural language processing and speech recognition. These automata are constructed in a way that conveniently solves the weighted version of the membership problem, that is, the problem of computing the weight of a string. In many cases, however, the WFA represents something like a "hypothesis space", where the weights represent some kind of desirability or quality. For example a natural language translation system may be implemented as a weighted transducer, which for an input string produces a WFA as output. This WFA then assigns weights to strings according to the likelihood that the string is a good translation of the original input. We may then wish to somehow enumerate a few of the "best" runs or strings with respect to a given WFA.

In a WFA over a semiring $\mathbb{S}$, transitions are assigned a weight in $\mathbb{S}$. Essentially, the weight of a run is the product of the weights of the edges traversed, and the weight of an input string is the sum of the weights of all its runs.[1] The problem of finding the best *runs* is well-explored: it is the problem of finding the shortest paths in a directed weighted graph. Notably, there are very efficient algorithms to, in order, enumerate the shortest paths through a graph, where the edge weights are usually interpreted as lengths and are, accordingly, summed up [2]. This corresponds to the use of the tropical semiring, whose multiplication is ordinary addition and whose addition takes the minimum. Algorithms for best runs in WFA over other types of semirings have also been investigated [6].

Not quite as well investigated is the k best strings problem (k-BSP) where the aim is to find the best *strings*, i.e., those with the least weight. This is different from the problem of finding the best runs if non-deterministic WFA are considered, as the

[1] Here, products and sums are built by using the multiplication and addition, respectively, of the semiring.

Martin Berglund, Frank Drewes: On the Complexity of Variants of the k Best Strings Problem, pp. 76–88 .
Proceedings of PSC 2010, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04597-8

weight of a string is the sum of the weights of all runs for that particular input string. Thus, while every run corresponds to a particular input string, the weight of the run does not in general coincide with the weight of the string. Furthermore, distinct runs may correspond to the same input string, whereas the k-BSP asks for the k best *unique* strings. This makes the problems differ even for the particular case of WFA over the tropical semiring, where the weight of an input string is always the lowest weight among all runs associated with the string. In the extreme case, the k best runs may all belong to the same input string.

In [7], an algorithm is presented that solves the k-BSP for WFA over the tropical semiring. This algorithm is based on a clever on-demand (or lazy) determinization, which stops when the k runs with the least weight (in the determinized part) have been found. The algorithm has been reported to be very efficient in practice. However, it appears to run in exponential time in some bad cases. Therefore, it is a natural question to ask whether there is a polynomial algorithm solving the problem.

In this paper, we give a positive answer to this question. Of course, this raises the question whether the setting can be generalized to other semirings without losing tractability. The first answers to this question will be given by considering a decision problem closely related to the k-BSP for $k = 1$, namely the *string quality threshold problem* (SQTP). Here, we are given a WFA and a threshold $t \in \mathbb{S}$, and the question is whether there exists a string whose weight is less than or equal to t (with respect to the order considered).

In summary, we establish the following three main results:

- The k-BSP for WFA over the tropical semiring is solvable in polynomial time.
- For WFA over the tropical semiring on pairs of numbers (which we call the complex tropical semiring), the SQTP is NP-complete.
- For WFA over the semiring of positive real numbers with the usual addition and multiplication, the SQTP is undecidable.

The remainder of the paper is structured as follows. In the next section, basic notions and notation are compiled. In Section 3, the problems to be investigated are defined. In Sections 4, 5, and 6, the three main results are shown. Finally, a short conclusion is given in Section 7.

2 Basic Notions and Notation

For $n \in \mathbb{N}$, we denote the set $\{1, \ldots, n\}$ by $[n]$. The set $\{x \mid x \in \mathbb{R}, x \geq 0\} \cup \{\infty\}$ is denoted by $\mathbb{R}_+^\infty$. Similarly, $\mathbb{C}_+^\infty$ denotes $\{x + yi \mid x \in \mathbb{R}_+^\infty, y \in \mathbb{R}_+^\infty\}$.

We denote a semiring as a tuple $(\mathbb{S}, \oplus, \otimes)$ where $\mathbb{S}$ is the domain, $\oplus$ the addition operator and $\otimes$ the multiplication operator. Semirings will often be equipped with a (possibly partial) order $\leq$, in which case the semiring is denoted by $(\mathbb{S}, \oplus, \otimes, \leq)$. If $\oplus$ and $\otimes$ (and $\leq$) are clear from the context, then the semiring may simply be denoted by $\mathbb{S}$.

For an alphabet Σ, Σ^* denotes the set of all strings over Σ. The empty string, i.e., the string of length 0, is denoted by ϵ. The length of a string s is denoted by $|s|$, and $s \cdot s'$ or simply ss' denotes the concatenation of s with another string s'. The notation $|S|$ is also used to denote the cardinality of a set S.

A weighted finite-state automaton (WFA) is a tuple $A = (\Sigma, Q, \mathbb{S}, \mu, \lambda, \rho)$ where Σ is a finite alphabet of input symbols, Q is a finite set of states, $\mathbb{S}$ is the semiring from which the weights are taken, $\mu \colon Q \times \Sigma \times Q \to \mathbb{S}$ is the weighted transition

function and $\lambda, \rho\colon Q \to \mathbb{S}$ are the initial and final weight vectors, respectively. The WFA A is *deterministic* if, for all $q \in Q$ and $a \in \Sigma$, there is at most one $q' \in Q$ such that $\mu(q, a, q') \neq 0$.

The transition function μ can alternatively be viewed as a set of rules, namely

$$R_\mu = \{q \xrightarrow{w,a} q' \mid (q, a, q') \in Q \times \Sigma \times Q \text{ and } \mu(q, a, q') = w \in \mathbb{S} \setminus \{0\}\}$$

(where 0 is the additive identity of $\mathbb{S}$). Thus, the case $\mu(q, a, q') = 0$ corresponds to a non-existing rule. We define the size $|A|$ of an automaton as the number of rules, i.e., $|A| = |R_\mu|$.

A WFA A computes a function $\underline{A}\colon \Sigma^* \to \mathbb{S}$, called a string series in the theory of weighted automata [1], as follows: for all strings $s = a_1 \cdots a_n$,

$$\underline{A}(s) = \sum_{p_1,\ldots,p_{n+1} \in Q} \lambda(p_1) \left(\prod_{i=1}^{n} \mu(p_i, a_i, p_{i+1}) \right) \rho(p_{n+1}).$$

Here, the sums and products are defined using the operators $\oplus$ and $\otimes$, resp., of the semiring. An alternating sequence of states and input symbols $p_1, a_1, p_2, \ldots, a_n, p_{n+1}$ is also called a run of A on s. The weight of the run is given by the product $\lambda(p_1) \left(\prod_{i=1}^{n} \mu(p_i, a_i, p_{i+1})\right) \rho(p_{n+1})$. In other words, $\underline{A}(s)$ is the sum of the weights of all runs on s.

By abuse of notation, $\underline{A}$ will simply be denoted by A from now on. We write $\mathrm{WFA}_\Sigma^{\mathbb{S}}$ to denote the set of all WFA over Σ and $\mathbb{S}$.

The reader may have noticed that we do not allow ϵ transitions in our WFA. However, this restriction is not essential in any case studied here. Its sole purpose is to simplify the presentation of the algorithm in Section 4. Let us have a look at an example semiring to wrap this section up.

Example 1 (The tropical semiring). The tropical semiring is an important case both for the following sections and in many practical applications. It is defined as $\mathsf{Trop} = (\mathbb{R}_+^\infty, \min, +, \leq)$, where min, + and $\leq$ all have their usual meanings on $\mathbb{R}_+^\infty$. Note that the product in Trop is ordinary addition. For all $A = (\Sigma, Q, \mathsf{Trop}, \mu, \lambda, \rho) \in \mathrm{WFA}_\Sigma^{\mathsf{Trop}}$ and all strings $s = a_1 \cdots a_n$ the formula above gives us

$$A(s) = \min_{p_1,\ldots,p_{n+1} \in Q} \lambda(p_1) + \left(\sum_{i=1}^{n} \mu(p_i, a_i, p_{i+1}) \right) + \rho(p_{n+1}).$$

That is, finding the weight that A assigns to the string s corresponds to finding the run on s with the minimal total weight. This makes the tropical semiring case closely related to various shortest-path problems, as we will see in coming sections.

Consider the input alphabet $\Sigma = \{a, b\}$. We give a WFA over Trop such that, for every $s \in \Sigma^*$, $A(s) = |s| - l$, where l is the length of the longest substring in s consisting only of the symbol a. For this, we let $Q = \{\text{prefix}, \text{middle}, \text{suffix}\}$, $\lambda(q) = 0$ for all $q \in Q$,

$$\begin{aligned} R_\mu = \{ &\text{prefix} \xrightarrow{1,x} \text{prefix } (x \in \Sigma), \\ &\text{prefix} \xrightarrow{1,b} \text{middle}, \\ &\text{middle} \xrightarrow{0,a} \text{middle}, \\ &\text{middle} \xrightarrow{1,b} \text{suffix}, \\ &\text{suffix} \xrightarrow{1,x} \text{suffix } (x \in \Sigma) \ \}, \end{aligned}$$

and

$$\rho(q) = \begin{cases} 0 & \text{if } q \in \{\text{middle}, \text{suffix}\} \\ \infty & \text{otherwise.} \end{cases}$$

Intuitively, the WFA guesses non-deterministically the part to be left out when counting the symbols the input string consists of.

3 Problem Definitions

As mentioned above, given a WFA $A \in \mathrm{WFA}^{\mathbb{S}}_{\Sigma}$ and some $k \in \mathbb{N}$, we are interested in computing the k "best" strings in the sense that these strings are assigned the lowest weights by A. The formal definition of the problem reads as follows.

Definition 2 (k best strings problem). *Let $(\mathbb{S}, \oplus, \otimes, \leq)$ be a partially ordered semiring, and let Σ be an alphabet. An instance of the* k best strings problem *(k-BSP) over $\mathbb{S}$ is a pair $(A, k) \in \mathrm{WFA}^{\mathbb{S}}_{\Sigma} \times \mathbb{N}$. A* solution *to the instance is a set S of strings in Σ^* such that $|S| = k$ and, for all strings s, s', if $A(s) < A(s')$ and $s' \in S$ then $s \in S$.*

The 1-BSP is the special case of the k-BSP where $k = 1$ is considered to be fixed.

Notice that the solution S is not necessarily unique, because each string $s' \in S$ can be replaced with any other string $s \in \Sigma^* \setminus S$ such that $A(s) = A(s')$. Thus, there may even be an infinite number of solutions. Also note that in some cases no solution may exist, because a solution cannot include elements from an infinite chain $s_0, s_1, s_2, \ldots$ such that $A(s_{i+1}) < A(s_i)$ for all $i \in \mathbb{N}$, i.e., the s_i get "better and better". This cannot happen if $<$ is well-founded, as will be the case in the next two sections.

We also consider a closely related decision problem.

Definition 3 (String quality threshold problem). *Let $(\mathbb{S}, \oplus, \otimes, \leq)$ be a partially ordered semiring, and let Σ be an alphabet. An instance of the* string quality threshold problem *(SQTP) is a pair $(A, t) \in \mathrm{WFA}^{\mathbb{S}}_{\Sigma} \times \mathbb{S}$. The question to be answered is whether there exists a string $s \in \Sigma^*$ such that $A(s) \leq t$.*

As usual, we shall identify a decision problem such as the SQTP with the set of all its *yes* instances. Thus, given an instance I, we write $I \in \mathrm{SQTP}$ to express that I is a *yes* instance of *SQTP*. Note that if the 1-BSP problem A has a solution $\{s\}$ we will have $(A, t) \in \mathrm{SQTP}$ for all $t \geq A(s)$.

The problems are closely related in the other direction as well: as long as $\leq$ is a total order, it holds that for all $(A, t) \in \mathrm{SQTP}$ all solutions $\{s\}$ to the 1-BSP problem $(A, 1)$ satisfy $A(s) \leq t$. That is, if we know that there exists some string with weight less than or equal to t then any algorithm solving the 1-BSP will have to find such a string.

4 A Polynomial k Best Strings Algorithm for the Tropical Case

In this section, we show that the k-BSP can be solved in polynomial time for the tropical semiring $\mathsf{Trop} = (\mathbb{R}^{\infty}_{+}, \min, +, \leq)$ (as defined in Example 1). For the rest of this section, let us consider an instance (A, k) of the k-BSP over Trop, where $A = (\Sigma, Q, \mathsf{Trop}, \mu, \lambda, \rho)$. Let us start with an important lemma.

Lemma 4 (Short minimal strings). *For any string $s \in \Sigma^*$ let $l = \left\lfloor \frac{|s|}{|Q|} \right\rfloor$. Then there exists at least l distinct strings $s_1, \ldots, s_l$ such that $|s_i| \leq |s|$ and $A(s_i) \leq A(s)$ for all $i \in [l]$.*

Proof. Let $s = a_1 \cdots a_n$. By the definition of $A(s)$, together with the fact that A is defined over Trop, we know that $A(s) = \min_{p_1,\ldots,p_{n+1} \in Q} \lambda(p_1) + (\sum_{i=1}^{n} \mu(p_i, a_i, p_{i+1})) + \rho(p_{n+1})$. Let $p_1, \ldots, p_{n+1} \in Q$ be one of the (not necessarily unique) choices of states that minimize the expression. Then, since $n + 1 > l|Q|$ there exists a state $q \in Q$ which occurs $l + 1$ or more times among $p_1, \ldots, p_{n+1}$ (by the pigeon hole principle). Let $i_1, \ldots, i_{l+1} \in [n]$ be the distinct indices such that $p_{i_j} = q$ for all j. For all $j \in [l+1]$ let

$$F(j) = \lambda(p_1) + \left(\sum_{h=1}^{i_j - 1} \mu(p_h, a_h, p_{h+1})\right) + \left(\sum_{h=i_{l+1}}^{n} \mu(p_h, a_h, p_{h+1})\right) + \rho(p_{n+1}).$$

Notice that $F(j) \leq A(s)$ for all $j \in [l+1]$, owing to the fact that all terms of the sum defining $F(j)$ are also part of the sum defining $A(s)$. Now, for each $j \in [l+1]$ construct the string $s'_j = a_1 \cdots a_{i_j - 1} a_{i_{l+1}} \cdots a_n$. Notice that these strings are pairwise distinct. For each of them it holds that $A(s'_j) \leq F(j)$, since $F(j)$ is the weight corresponding to one of the possible state choices for the minimization in the evaluation of $A(s'_j)$.

Thus, as required, we have obtained l distinct strings $s'_1, \ldots, s'_l$ such that $A(s'_j) \leq F(j) \leq A(s)$. □

As a rather direct consequence, we get the following.

Corollary 5 (Existence of a short k-BSP solution). *If (A, k) has a solution, then it has a solution S such that $|s| \leq k|Q|$ for all $s \in S$.*

To see this, simply consider a solution S that contains a string s longer than the corollary states is necessary. By Lemma 4, there exists a shorter string $s' \notin S$ of the same weight, which s can be replaced with.

Now we are ready to describe, as a first step towards solving the k-BSP, an algorithm that solves the 1-BSP. This does in the process solve the SQTP, since our semiring is totally ordered. For solving the 1-BSP, we can simply apply Dijkstra's algorithm to A.

Algorithm 6 (1-BSP and SQTP algorithm). View A as a directed labeled weighted graph by considering the states to be nodes and the rules to be weighted edges. Then simply apply Dijkstra's algorithm [8] to A in the following way:

1. For each $q \in Q$ the algorithm assigns a weight $\mathit{weight}(q)$ to q. Initially, $\mathit{weight}(q) = \lambda(q)$ and $U = Q$.
2. Take any $q \in U$ with $\mathit{weight}(q) = \min_{q' \in U} \mathit{weight}(q')$.
3. For all edges $q \xrightarrow{w,a} q'$ set $\mathit{weight}(q') = \min(\mathit{weight}(q'), \mathit{weight}(q) + w)$.
4. Let $U = U \setminus \{q\}$. If $U \neq \emptyset$ go to step 2.

Now create the directed labeled graph $G = (V, E)$ where $V = Q$ and $E = \{q \xrightarrow{a} q' \mid (q, a, q', w) \in \mu, \mathit{weight}(q') = \mathit{weight}(q) + w\}$. Then let $\hat{w} = \min_{q \in Q} \mathit{weight}(q) + \rho(q)$, let $F = \{q \in Q \mid \mathit{weight}(q) + \rho(q) = \hat{w}\}$, and let $I = \{q \in Q \mid \mathit{weight}(q) = \lambda(q)\}$. Then simply perform a breadth-first search to find the (not necessarily unique) shortest path $q_1 \xrightarrow{a_1} \ldots \xrightarrow{a_n} q_{n+1}$ through G such that $q_1 \in I$ and $q_{n+1} \in F$.

Now, $s = a_1 \cdots a_n$ is a solution to the 1-BSP, and we have $A(s) = \hat{w}$. Consequently, $(A, t) \in$ SQTP if and only if $t \geq \hat{w}$. The running time of this algorithm,

being dominated by the running time of Dijkstra's algorithm, is $\mathcal{O}(|A| + |Q| \log |Q|)$, provided that some well-known optimizations are made [8].

The following lemma summarizes the properties of Algorithm 6.

Lemma 7. *Algorithm 6 solves the* 1*-BSP in time* $\mathcal{O}(|A| + |Q| \log |Q|)$*. Furthermore, if it returns* $\{s\}$*, then* $|s| \leq |s'|$ *holds for every other solution* $\{s'\}$ *to the* 1*-BSP instance.*

The fact that the lemma above ensures $|s| \leq |s'|$ will enable us to exploit Corollary 5. As building blocks for the general algorithm, let us make two more definitions.

Definition 8 (*S*-complement WFA**).** *For any finite set* $S \subset \mathcal{P}(\Sigma^*)$ *let* $A_S \in \mathrm{WFA}_\Sigma^{\mathsf{Trop}}$ *denote the automaton* $(\Sigma, Q_S, \mathsf{Trop}, \mu_S, \lambda_S, \rho_S)$*, constructed in the following way.*

- $Q_S = \{sink, r_\epsilon\} \cup \{r_s \mid s$ *is a prefix of a string in* $S\}$*.*
- *For all* $q \in Q_S$*,*

$$\lambda(q) = \begin{cases} 0 & \text{when } q = r_\epsilon \\ \infty & \text{otherwise,} \end{cases}$$

$$\rho(q) = \begin{cases} \infty & \text{if } q = r_s \text{ for some } s \in S \\ 0 & \text{otherwise.} \end{cases}$$

- *For all* $r_s \in Q_S$ *and all* $c \in \Sigma$*,* R_{μ_S} *contains the rules*
  - $r_s \xrightarrow{0,c} r_{sc}$ *if* $r_{sc} \in Q_S$*,*
  - $r_s \xrightarrow{0,c} sink$ *if* $r_{sc} \notin Q_S$*, and*
  - $sink \xrightarrow{0,c} sink$*.*

  The reader should easily be able to check that, for all $s \in \Sigma^*$,

$$A_S(s) = \begin{cases} \infty \text{ if } s \in S \\ 0 \quad \text{otherwise.} \end{cases}$$

Definition 9 (Product-$\mathrm{WFA}^{\mathsf{Trop}}$**).** *For all WFA* $A_1 = (\Sigma, Q_1, \mathsf{Trop}, \mu_1, \lambda_1, \rho_1)$ *and* $A_2 = (\Sigma, Q_2, \mathsf{Trop}, \mu_2, \lambda_2, \rho_2)$*, let* $A_1 \times A_2$ *denote the* product automaton*, defined by* $A_1 \times A_2 = (\Sigma, Q_1 \times Q_2, \mathsf{Trop}, \mu, \lambda, \rho)$ *where, for all* $(q_1, q_2) \in Q_1 \times Q_2$*,*

- $\lambda(q_1, q_2) = \lambda_1(q_1) + \lambda_2(q_2)$*,*
- $\rho(q_1, q_2) = \rho_1(q_1) + \rho_2(q_2)$*, and*
- $\mu((q_1, q_2), a, (q_1', q_2')) = \mu_1(q_1, a, q_1') + \mu_2(q_2, a, q_2')$ *for all* $(q_1', q_2') \in Q_1 \times Q_2$ *and all* $a \in \Sigma$*.*

It should be clear that, for all $s \in \Sigma^*$, we have $(A_1 \times A_2)(s) = A_1(s) + A_2(s)$. Given the 1-BSP algorithm, it is now straightforward to construct the algorithm that solves the k-BSP.

Algorithm 10 (*k*-BSP algorithm). To compute a solution to the k-BSP instance (A, k), we proceed as follows.

1. Initially, let $S = \emptyset$.
2. If $|S| = k$ halt and return S as the answer.

3. Construct the automaton $A' = A \times A_S$ where A_S is the S-complement WFA as in Definition 8 (this gives $A' = A$ for $S = \emptyset$).
4. Apply Algorithm 6 to the 1-BSP instance A', and let s be the answer the algorithm computes.
5. Let $S = S \cup \{s\}$ and go to step 2.

There is one degenerate case, where the string s computed in step 4 satisfies $A'(s) = \infty$ (that is, all strings that have a finite weight in A have already been picked). To handle that edge case simply pick the remaining $|S| - k$ strings from $\Sigma^* \setminus S$ arbitrarily and halt. In all other cases, $s \notin S$ will hold at step 3.

Next, we establish the correctness and complexity of Algorithm 10 to complete this section.

Theorem 11. *If applied to a k-BSP instance (A, k), Algorithm 10 returns a correct solution in time $\mathcal{O}((k^3|A|^2)\log(k|A|))$.*

Proof. The correctness of the algorithm is straightforward to show. Consider steps 3 and 4 of the algorithm, and suppose that $A'(s) \neq \infty$. By the properties of A' noted above, and by Lemma 7, s is a shortest string such that $A(s) = \min\{A(s') \mid s' \in \Sigma^* \setminus S\}$. By induction, this means that the set S that is eventually returned is a valid solution. Furthermore, S is a shortest solution, i.e., every other solution S' satisfies $\sum_{s\in S'} |s| \geq \sum_{s\in S} |s|$.

As for the complexity of the algorithm, consider the k^{th} iteration. Lemma 4 and the fact that S is a shortest solution yield $\max_{s\in S} |s| \leq k|Q|$. This means that $\sum_{s\in S} |s| \in \mathcal{O}(k^2|Q|)$. Constructing the automaton A_S according to Definition 8 will then give us $\mathcal{O}(k^2|Q|)$ states, and since A_S is deterministic we have $|A_S| \in \mathcal{O}(k^2|Q|)$.[2] Thus, A' consists of $\mathcal{O}(k^2|Q|^2)$ states and $\mathcal{O}(k^2|Q||A|)$ rules. By Lemma 7, this means that Algorithm 6 runs in time $\mathcal{O}(k^2|A||Q| + (k^2|Q|^2)\log(k^2|Q|^2))$. Summing up over the k iterations of the algorithm, this yields a running time of

$$\mathcal{O}(k^3|A||Q| + k^3|Q|^2\log(k^2|Q|^2)) = \mathcal{O}(k^3|A||Q| + k^3|Q|^2\log(k|Q|)).$$

Since $|Q| < |A|$ in all non-degenerate cases, this yields the bound stated. □

5 The Complex Tropical Case is NP-Complete

We now consider the extension of the tropical semiring to the plane, called the complex tropical semiring. It is defined as $\mathsf{Trop}^2 = (\mathbb{C}_+^\infty, \min, +, \leq)$ where min and + are component-wise minimum and addition (i.e., $\min(x + yi, x' + y'i) = \min(x, x') + \min(y, y')i$, and similarly for +). The (partial) order $\leq$ of this semiring is also defined component-wise, i.e., for all $a, b \in \mathbb{C}_+^\infty$ we have $a \leq b$ if and only if $a = \min(a, b)$. This case is an interesting extension of the normal tropical case, and would be useful in settings where one wishes to track multiple qualities of a string independently. For example in the case of natural language correction one could let the first component signify the prevalence of typing mistakes (spelling fixes), whereas the second component could signify the severity of structural mistakes (for example typical mistakes for non-native speakers, like verb-subject agreement). We prove that the SQTP is NP-complete even for deterministic WFA over Trop^2. We start by showing that the problem is in NP, followed by showing that it is NP-hard.

[2] Here, we consider $|\Sigma|$ to be a constant.

Lemma 12. *The SQTP for* WFA *over* Trop^2 *is in NP.*

Proof. Lemma 4 holds even for Trop^2, with precisely the same proof. As a direct consequence Corollary 5 also holds. From this it follows that for any WFA $A = (\Sigma, Q, \mathsf{Trop}^2, \mu, \lambda, \rho)$ and $t \in \mathbb{C}_+^\infty$ such that $(A, t) \in$ SQTP there exists some $s \in \Sigma^*$ with $|s| \leq |Q|$ such that $A(s) \leq t$.

We can then solve the SQTP instance (A, t) by non-deterministically choosing any string $s \in \Sigma^*$ with $|s| \leq |Q|$ and checking if $A(s) \leq t$. If this succeeds then $(A, t) \in$ SQTP. □

Lemma 13. *The SQTP for deterministic* WFA *over* Trop^2 *is NP-hard .*

To prove the theorem by reduction, recall the shortest weight-constrained path problem [3].

Definition 14 (Shortest weight-constrained path). *An instance of the shortest weight-constrained path problem (SWCP) is a tuple* $I = (G, w, l, (u, v), (M_w, M_l))$, *where* G *is a directed graph* $G = (V, E)$, $w \colon E \to \mathbb{N}^+$ *is a weight function,* $l \colon E \to \mathbb{N}^+$ *is a length function,* $u, v \in V$ *and* $M_w, M_l \in \mathbb{N}^+$. *The question to be answered is whether there exists a path from* u *to* v *such that the total weight of all edges on the path is less than* M_w *and the total length of all edges on the path is less than* M_l.

This problem is known to be NP-complete [3]. Given an instance I of the SWCP as above, we construct a weighted automaton A over Trop^2 such that $(A, t) \in$ SQTP for $t = M_w + M_l i$ if and only if $I \in$ SWCP. The construction is straightforward, as follows.

Construct the WFA $A = (\Sigma, Q, \mathsf{Trop}^2, \mu, \lambda, \rho)$ as follows. Let $Q = V$, let $\Sigma = \{\widehat{v_1 v_2} \mid (v_1, v_2) \in E\}$, for all $q \in Q$ let

$$\lambda(q) = \begin{cases} 0 + 0i & \text{if } q = u \\ \infty + \infty i & \text{otherwise,} \end{cases}$$

and

$$\rho(q) = \begin{cases} 1 + 0i & \text{if } q = v \\ 0 + 0i & \text{otherwise.} \end{cases}$$

Then we simply let R_μ consist of all rules $v_1 \xrightarrow{\hat{w}, \widehat{v_1 v_2}} v_2$, such that $(v_1, v_2) \in E$ and $\hat{w} = w(v_1, v_2) + l(v_1, v_2)i$.

Thus, the WFA interprets the input string as a sequence of edges in G. If this sequence is a path starting at u, the state corresponding to the node reached on this path will carry the weight equal to the weight-length combination up to that point; all other states will carry the weight $\infty + \infty i$. Owing to the choice of ρ, this yields the desired result, i.e., $(A, t) \in$ SQTP for $t = M_w + M_l i$ if and only if $I \in$ SWCP.

Summing up, we have proved the following theorem.

Theorem 15. *The SQTP for WFA over* Trop^2 *is NP-complete.*

Proof. Follows from Lemma 12 together with Lemma 13. □

6 The General Case is Undecidable

We finally identify a semiring for which the SQTP turns out to be undecidable: the semiring $(\mathbb{R}_+, +, *, \leq)$, where $+$ and $*$ are ordinary addition and multiplication and $\leq$ is the usual order on $\mathbb{R}_+$. This case is not in itself practically motivated, but it is very useful to clearly illustrate that there is no hope for a truly general solution to the SQTP for arbitrary semirings.

For a proof by reduction, we consider the problem whether a Turing machine accepts the empty string (or, equivalently, whether a Turing machine without input halts). We reduce this problem to the SQTP for WFA over $\mathbb{R}_+$, constructing the WFA in such a way that the string series it computes will assign the weight 1 to some string only if the Turing machine has an accepting run. Otherwise, the weights of all strings will be strictly larger than 1. Throughout this section non-determinism, in the same sense as in non-deterministic finite automata, will be a key concern, importantly we will use non-deterministic Turing machines [4] as the starting point of the reduction. That is, during the computation the Turing machine will sometimes make non-deterministic choices between different instructions to jump to. As usual, the acceptance criterion is that a computation ending in the accepting state exists. The WFA constructed by the reduction will be non-deterministic as well.

Let us start by defining the precise machine model we will use. Rather than using ordinary Turing machines, we use the well-known two-counter machines [5] as our starting point. Let us first recall the basic definition of the original two-counter machine.

Definition 16 (Two-counter machine). *A two-counter machine without input is a tuple $M = (C, P, c_0)$ consisting of a finite set C of states, a starting state $c_0 \in C$, and a program $P\colon C \to (\{\mathit{inc}_1, \mathit{inc}_2\} \times C) \cup (\{\mathit{jzdec}_1, \mathit{jzdec}_2\} \times C \times C) \cup \{\mathit{accept}\}$.*

The semantics of a two-counter machine is the usual one from [5]. The machine starts in state c_0 with the counters set to zero. In state c, the instruction $P(c)$ is executed:

1. $(\texttt{inc}_i, c')$ increments counter i and continues in state c',
2. $(\texttt{jzdec}_i, c', c'')$ continues in state c' if counter i is zero, and decrements the counter and continues in state c'' if it is not zero, and
3. `accept` halts and accepts the input.

We now adjust this into another type of Turing machine which is equivalent but more convenient for our purpose. The adjustment consists in

- adding another two counters (used for temporary "scratch" values) and allowing all counters to contain negative values, and
- breaking the `jzdec` instruction into two, a `zero` instruction which simply makes the computation immediately fail if the counter tested is non-zero, and a `jump` instruction which *non-deterministically* chooses where to jump. That is, the `jump` instruction has two targets and the machine non-deterministically picks one of them.

This adjustment does not restrict the computational power of counter machines. We provide the definition here for convenience. Four counters are not strictly needed, but are convenient to let us quickly sketch how the `jzdec` instruction can be simulated using the `zero` and `jump` instructions, demonstrating equivalence.

Definition 17 (Four-counter machine). *A non-deterministic four-counter machine is a triple $M = (C, P, c_0)$ consisting of a finite set C of states, a starting state $c_0 \in C$, and a program*

$$P\colon C \to (\bigcup_{i\in[4]} \{\mathit{inc}_i, \mathit{dec}_i, \mathit{zero}_i\} \times C) \cup (\{\mathit{jump}\} \times C \times C) \cup \{\mathit{accept}\}.$$

The computation starts in state c_0 with all four counters set to zero. The semantics of the instructions is given as follows, for all $i \in [4]$:

- inc_i *increments the counter i,*
- dec_i *decrements the counter i (which may result in negative values),*
- zero_i *makes the computation immediately fail if the counter i is not zero,*
- jump *non-deterministically jumps to one of the two states given in the instruction, and*
- $\mathtt{accept}$ *halts and accepts.*

Such a machine is computationally equivalent to a two-counter machine. There is a straightforward simulation of a two-counter machine by a four-counter machine. The latter mimics the two-counter behavior in counters 1 and 2 while using counters 3 and 4 as temporary "scratch" variables. As a building block we can, using counter 4 as a temporary variable, implement the macro instruction $\mathtt{copy}_{i\to 3}$ (for $i \in [2]$), which sets the value of counter 3 equal to the (non-negative) value in counter i. Assume that we have counters 3 and 4 set to zero, then transfer the value in counter i into both counter 3 and counter 4 by running the sequence $\mathtt{dec}_i$, $\mathtt{inc}_3$, $\mathtt{inc}_4$ in a loop until counter i is zero (simply loop non-deterministically many times and execute $\mathtt{zero}_i$ when the loop ends). Finish the procedure by transferring the contents of counter 4 back into counter i in the same way. Next consider the two-counter machine instruction $P(c) = (\mathtt{jzdec}_1, c', c'')$, as given in to Definition 16. This can be translated into the following instructions (using pairwise distinct new states q_x):

$$\begin{aligned}
P(c) &= (\mathtt{jump}, q_{zero1}, q_{nonzero1}),\\
P(q_{zero1}) &= (\mathtt{zero}_1, c'),\\
P(q_{nonzero1}) &= (\mathtt{dec}_1, q_{copy3}),\\
P(q_{copy3}) &= (\mathtt{copy}_{1\to 3}, q_{test3}),\\
P(q_{test3}) &= (\mathtt{jump}, q_{end}, q_{dec3}),\\
P(q_{dec3}) &= (\mathtt{dec}_3, q_{loop}),\\
P(q_{loop}) &= (\mathtt{jump}, q_{test3}, q_{test3}),\\
P(q_{end}) &= (\mathtt{zero}_3, c'').
\end{aligned}$$

Notice that if this scheme of translating a two-counter machine is used, no counter in an accepting computation will ever actually become negative, since the above snippet will immediately run into an infinite loop if it ever produces a negative counter (by making the wrong non-deterministic choice in the first line). Note that this shows that the accepting run of the four-counter machine simulating a two-counter machine is uniquely determined if it exists. In fact, this holds for arbitrary starting configurations. In the following, we assume that the four-counter machines we are dealing with have this property. For our reduction, it is useful to define its unique accepting computation (if it accepts), called the *trace* of the machine.

Definition 18 (Run of a four-counter machine M). *Let $M = (C, P, c_0)$ be a four-counter machine. The* trace alphabet *of M is*

$$\begin{aligned}\Sigma(M) = \{&\mathit{jump}_{c_i}^{[c]} \mid c \in C, P(c) = (\mathit{jump}, c_1, c_2), i \in [2]\} \cup \\ &\{o_i^{[c]} \mid c \in C, o \in \{\mathit{inc}, \mathit{dec}, \mathit{zero}\}, i \in [4], P(c) = (o_i, c') \text{ for a } c' \in C\} \cup \\ &\{\mathit{accept}^{[c]} \mid c \in C, P(c) = \mathit{accept}\}.\end{aligned}$$

The trace of M, *starting in state c with counter values $\kappa_1, \ldots, \kappa_4 \in \mathbb{N}$, is denoted by $trace(M, c, \kappa_1, \ldots, \kappa_4)$. It is the string $r \in \Sigma(M)^*$ defined as follows:*

1. *If $P(c) = \mathit{accept}$, then $r = \mathit{accept}^{[c]}$.*
2. *If $P(c) = (\mathit{inc}_i, c')$, then the trace is $\mathit{inc}_i^{[c]} \cdot trace(M, c', \kappa_1', \ldots, \kappa_4')$, where $\kappa_i' = \kappa_i + 1$ and $\kappa_j' = \kappa_j$ for all $j \neq i$.*
3. *If $P(c) = (\mathit{dec}_i, c')$, then the trace is $\mathit{dec}_i^{[c]} \cdot trace(M, c', \kappa_1', \ldots, \kappa_4')$, where $\kappa_i' = \kappa_i - 1$ and $\kappa_j' = \kappa_j$ for all $j \neq i$.*
4. *If $P(c) = (\mathit{zero}_i, c')$ then r is undefined unless $\kappa_i = 0$, in which case $r = \mathit{zero}_i^{[c]} \cdot trace(M, c', \kappa_1, \ldots, \kappa_4)$.*
5. *If $P(c) = (\mathit{jump}, c', c'')$ then $r = \mathit{jump}_{c'}^{[c]} \cdot trace(M, c', \kappa_1, \ldots, \kappa_4)$ or $r = \mathit{jump}_{c''}^{[c]} \cdot trace(M, c'', \kappa_1, \ldots, \kappa_4)$, depending on which one is defined. If neither of them is defined then r is undefined.*

Notice that $trace(M, c_0, 0, \ldots, 0)$ is defined if and only if the machine accpets.

With this out of the way we get to the core part of this section. The following construction will take any four-counter machine M and construct a WFA $A \in \mathrm{WFA}_{\Sigma(M)}^{\mathbb{R}_+}$ such that $A(s) \leq 1$ if and only if s is a valid trace of M, that is $s = trace(M, c_0, 0, \ldots, 0)$.

Algorithm 19 (Four-counter WFA reduction). Let $M = (C, P, c_0)$ be a four-counter machine as above. We construct $A = (\Sigma, Q, \mathbb{R}_+, \mu, \lambda, \rho)$ such that there exists $s \in \Sigma^*$ with $A(s) \leq 1$ if and only if $trace(M, c_0, 0, \ldots, 0)$ is defined. This construction can be performed in the following way. Let $Q = \{p_c \mid c \in C\} \cup \bigcup_{i \in [4]} \{c_{i,\mathrm{up}}, c_{i,\mathrm{down}}\} \cup \{\mathrm{fail}, \mathrm{final}\}$. Let $\Sigma = \Sigma(M)$ be as in Definition 18. For all $q \in Q$, let

$$\lambda(q) = \begin{cases} 1 \text{ for } q \in \{p_{c_0}, \mathrm{fail}\} \cup \{c_{i,\mathrm{up}}, c_{i,\mathrm{down}} \mid i \in [4]\} \\ 0 \text{ otherwise} \end{cases}$$

and

$$\rho(q) = \begin{cases} 1 \text{ if } q \in \{\mathrm{fail}\} \cup \{p_c \mid c \in C\} \\ 0 \text{ otherwise.} \end{cases}$$

Of course, the trick lies in the way in which R_μ is constructed. Each state carries a weight holding some invariant meaning in each step over a string. We start with some rules that will keep weights constant in certain situations (i.e., the rules are loops with a weight of 1, the neutral element with respect to multiplication):

C1. $\{\mathrm{fail} \xrightarrow{1,x} \mathrm{fail} \mid x \in \Sigma \setminus \{\mathtt{zero}_i^{[c]} \mid i \in [4], c \in C\}\}$
C2. $\{c_{i,d} \xrightarrow{1,x} c_{i,d} \mid d \in \{\mathrm{up}, \mathrm{down}\}, i \in [4], x \in \Sigma \setminus \{\mathtt{inc}_i^{[c]}, \mathtt{dec}_i^{[c]} \mid c \in C\}\}$

Next, we define rules to manage the accepting state:

S1. $\{\mathrm{fail} \xrightarrow{1,\mathtt{accept}^{[c]}} \mathrm{final} \mid c \in C\}$
S2. $\{\mathrm{final} \xrightarrow{1,x} \mathrm{fail} \mid x \in \Sigma\}$

The following rules are for managing the program counter:

P1. $\{p_c \xrightarrow{1,o_i^{[c]}} p_{c'} \mid c, c' \in C, i \in [4], o \in \{\texttt{inc}, \texttt{dec}, \texttt{zero}\}, (o_i, c') = P(c)\}$
P2. $\{p_c \xrightarrow{1,\texttt{jump}_{c'}^{[c]}} p_{c'} \mid (\texttt{jump}, c_1, c_2) = P(c), c' \in \{c_1, c_2\}\}$
P3. $\{p_{c'} \xrightarrow{1,x^{[c]}} \text{fail} \mid c, c' \in C, x^{[c]} \in \Sigma, c \neq c'\}$

Some rules are needed to manage the counters:

N1. $\{c_{i,\text{up}} \xrightarrow{2,\texttt{inc}_i^{[c]}} c_{i,\text{up}} \mid i \in [4], c \in C\}$
N2. $\{c_{i,\text{down}} \xrightarrow{1/2,\texttt{inc}_i^{[c]}} c_{i,\text{down}} \mid i \in [4], c \in C\}$
N3. $\{c_{i,\text{up}} \xrightarrow{1/2,\texttt{dec}_i^{[c]}} c_{i,\text{up}} \mid i \in [4], c \in C\}$
N4. $\{c_{i,\text{down}} \xrightarrow{2,\texttt{dec}_i^{[c]}} c_{i,\text{down}} \mid i \in [4], c \in C\}$

Finally, the following rules implement the `zero` instruction:

Z1. $\{c_{i,\text{up}} \xrightarrow{1/3,\texttt{zero}_i^{[c]}} \text{fail} \mid i \in [4], c \in C\}$
Z2. $\{c_{i,\text{down}} \xrightarrow{1/3,\texttt{zero}_i^{[c]}} \text{fail} \mid i \in [4], c \in C\}$
Z3. $\{\text{fail} \xrightarrow{1/3,\texttt{zero}_i^{[c]}} \text{fail} \mid i \in [4], c \in C\}$

The idea is that the weight of the trace $\mathit{trace}(M, c_0, 0, 0, 0, 0)$, if it exists, is 1, while all other strings over Σ will get a weight strictly greater than 1. This algorithm may take some explanation to be convincing. Let us note the key invariants exhibited by the construction.

The weight carried by the states $c_{i,\text{up}}$ and $c_{i,\text{down}}$ ($i \in [4]$) will always be 2^x and 2^{-x}, respectively, for some $x \in \mathbb{Z}$. In fact, the rule schemas N1–N4 ensure that the value x corresponds exactly to the value that the counter i would have at the corresponding point in the computation of M. In this way, the counters are represented.

The states p_c represent the current state of M. At each point in time, there is exactly one such state with weight 1, all others carrying the weight 0. Rule schema P3 ensures that if operations ever occur in an order that is impossible in M, weight will be added to the state "fail". This enforces program flow.

This brings us to the important "fail" state, which starts out with the weight 1 and will, by rule schema C1, always keep its weight from the previous step for all input symbols except `zero`. The symbol `zero` is handled specially by the rule schemas Z1–Z3. Let f be the weight of "fail" when encountering the symbol $\texttt{zero}_i^{[c]}$. Since the counter states will have weights 2^x and 2^{-x} (for some $x \in \mathbb{Z}$) the state "fail" gets assigned the weight $\frac{1}{3}f + \frac{1}{3}2^x + \frac{1}{3}2^{-x}$. Notice that if $f = 1$ this sum will be equal to 1 if and only if $x = 0$, and if $f > 1$ then the sum will always be greater than 1. This means that the state "fail" will carry the weight 1 if it did so before and the counter i is zero. Otherwise, it will carry a weight strictly larger than 1. This encodes the effect of the instruction $\texttt{zero}_i$.

Notice that the state "final" gets a weight greater than or equal to 1 when $\texttt{accept}^{[c]}$ is encountered (rule schema S1). On all symbols the weight of "final" gets added to "fail" (rule schema S2), which means that if `accept` is encountered in any but the last position this forces the weight of "fail" to be greater than 1. This enforces a valid end state.

Finally, ρ sums up the weights of "fail" and all the p states. The p states get set to 0 when encountering the right $\texttt{accept}^{[c]}$, so all that remains is the "fail" state weight. As we have seen, "fail" will maintained equal to 1 if all steps follow the constraints of M, and will end up greater than 1 otherwise.

Let us wrap this section up with the theorem stating the result of the reduction.

Theorem 20. *The SQTP is undecidable for WFA over* $(\mathbb{R}_+, +, *, \leq)$.

Proof. Algorithm 19 converts a non-deterministic four-counter machine M into a WFA A over $\mathbb{R}_+$ such that $(A, 1) \in$ SQTP if and only if M accepts (which is an undecidable problem). □

7 Conclusions

We have shown that the k-BSP is efficiently computable in the tropical case by a straightforward algorithm, while even the SQTP is difficult in the complex tropical case and undecidable for the positive real numbers.

Some remaining tasks for future work include improving the polynomial bound for the tropical k-BSP algorithm, which is likely to be possible since the worst-cases of the different parts of the algorithm almost seem to be mutually exclusive. In fact, it may also be worthwhile to analyze the situations in which the algorithm in [7] may exhibit an exponential worst-case behaviour. The algorithm uses a priority queue to determine in which direction the determinization algorithm should proceed. In some bad cases, the problem seems to be that this priority queue may not contain enough information in order for the strategy to become efficient. Hence, one could try to find a refined definition of priorities that (provably) avoids the problem.

Of course, there are many other semirings left for which polynomial solutions of the k-BSP may be obtainable. Moreover, one could try to abstract from concrete semirings by studying properties that give rise to polynomial solutions.

References

1. M. Droste, W. Kuich, and H. Vogler: *Handbook of Weighted Automata*, Springer Publishing Company, Incorporated, 2009.
2. D. Eppstein: *Finding the k shortest paths.* SIAM J. Comput., 28(2) 1999, pp. 652–673.
3. M. R. Garey and D. S. Johnson: *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1990.
4. J. E. Hopcroft, R. Motwani, and J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, Pearson Education International, Upper Saddle River, N.J. 04758, 2003.
5. M. L. Minsky: *Computation: finite and infinite machines*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.
6. M. Mohri: *Semiring frameworks and algorithms for shortest-distance problems.* J. Autom. Lang. Comb., 7(3) 2002, pp. 321–350.
7. M. Mohri and M. Riley: *An efficient algorithm for the n-best-strings problem*, in In Proceedings of the International Conference on Spoken Language Processing 2002, 2002.
8. M. A. Weiss: *Data structures and algorithm analysis*, Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1992.