Accepted Manuscript

Parameterized Searching with mismatches for run-length encoded strings

Alberto Apostolico, Péter L. Erdős, Alpár Jüttner

PII: S0304-3975(12)00238-1
DOI: 10.1016/j.tcs.2012.03.018
Reference: TCS 8800

To appear in: *Theoretical Computer Science*

Please cite this article as: A. Apostolico, P.L. Erdős, A. Jüttner, Parameterized Searching with mismatches for run-length encoded strings, *Theoretical Computer Science* (2012), doi:10.1016/j.tcs.2012.03.018

Parameterized Searching with Mismatches for Run-length Encoded Strings☆

Alberto Apostolico[1]

College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30318, USA and *Dipartimento di Ingegneria dell' Informazione, Università di Padova Padova, Via Gradenigo 6/A, 35131 Padova, Italy,*

Péter L. Erdős[2]

A. Rényi Institute of Mathematics, Hungarian Academy of Sciences, Budapest, P.O. Box 127, H-1364 Hungary

Alpár Jüttner[3]

Department of Operations Research, Eötvös University of Sciences, Pázmány Péter sétány 1/C, Budapest, H-1117 Hungary

Abstract

Parameterized matching between two strings occurs when it is possible to reduce the first one to the second by a renaming of the alphabet symbols. We present an algorithm for searching for parameterized occurrences of a patten in a textstring when both are given in run-length encoded form. The proposed method extends to alphabets of arbitrary yet constant size with $O(|r_p| \times |r_t|)$ time bounds, previously achieved only with binary alphabets. Here r_p and r_t denote the number of runs in the corresponding encodings for p and t. For general alphabets, the time bound obtained by the present method exhibits a polynomial dependency on the alphabet size. Such a performance is better than applying convolution to the cleartext, but leaves the problem still open of designing an alphabet independent $O(|r_p| \times |r_t|)$ time algorithm for this problem.

Keywords: string searching, parameterized matching, bipartite graphs, parametric graph matching

1. Introduction

String searching is one of the basic primitives of computation. In the standard formulation of the problem, we are given a pattern and a text and it is required to find all occurrences of the pattern in the text. Several variants of the problem have also been considered, e.g., allowing mismatches, insertions, deletions, swaps etc. In the parameterized variant, a match exists at one position of the text if the alphabets of pattern and text can be consistently mapped into one another in such a way that all characters match pairwise.

☆An extended abstract related to this work was presented at the SPIRE 2010 Symposium and is reported in its Proceedings.

Email addresses: `axa@cc.gatech.edu` (Alberto Apostolico), `elp@renyi.hu` (Péter L. Erdős), `alpar@cs.elte.hu` (Alpár Jüttner)

[1]Work carried out in part while visiting P.L. Erdős at the Rényi Institute, with support from the Hungarian Bioinformatics MTKD-CT-2006-042794, Marie Curie Host Fellowships for Transfer of Knowledge. Additional partial support was provided by the United States - Israel Binational Science Foundation (BSF) Grant No. 2008217, and by the Research Program of Georgia Tech.

[2]This work was supported in part by Alexander von Humboldt Foundation and by the Hungarian NSF, under contract NK 78439 and K 68262.

[3]This work was supported by OTKA grant CK80124.

More formally, two strings $\mathbf{y}$ and $\mathbf{y'}$ of equal length over respective alphabets Σ_y and $\Sigma_{y'}$ are said to *parameterized match* if there exists a bijection $\pi : \Sigma_y \to \Sigma_{y'}$ such that $\pi(\mathbf{y}) = \mathbf{y'}$, i.e., renaming each character of $\mathbf{y}$ according to its corresponding element under π yields $\mathbf{y'}$. (Here, for simplicity, we assume that all symbols of both alphabets are used somewhere.) Two natural problems are then *parameterized matching*, which consists of finding all positions of some text $\mathbf{x}$ where a pattern $\mathbf{y}$ parameterized matches a substring of $\mathbf{x}$, and *approximate parameterized matching*, which seeks, at each location of $\mathbf{x}$, a bijection π maximizing the number of parameterized matches at that location.

The first variant was introduced and studied by B. Baker [2, 3] and others, motivated by issues of program compaction in software engineering. In [2, 3], optimal, linear time algorithms were given under the assumption of constant size alphabets. A tight bound for the case of an alphabet of unbounded sizes was later presented in [1].

In this paper we study approximate variants of the problem where a (possibly controlled) number of mismatches is allowed. Hence, we are concerned with the second variant. Formally, we seek to find, for given text $\mathbf{x} = x_1 x_2 \dots x_n$ and pattern $\mathbf{y} = y_1 y_2 \dots y_m$ over respective alphabets Σ_t and Σ_p, the injection π_i from Σ_p to Σ_t maximizing the number of matches between $\pi_i(\mathbf{y})$ and $x_i x_{i+1} \dots x_{i+m-1}$ (for all $i = 1, 2, \dots n - m + 1$).

The general version of the problem can be solved in time $O\left(nm(\sqrt{m} + \log n)\right)$ by reduction to bipartite graph matching (refer to, e.g., [4]): each mutual alignment defines one graph in which edges are weighed according to the number of effacing characters and the problem is to choose the set of edges of maximum weight. Note that for fixed alphabet sizes the number of possible injections is also finite and thus it is enough to try them out individually through resort to convolution, resulting in total $O(n \log n)$ time overall. This no longer appears to be possible as soon as one of the alphabets is unbounded.

In [5], the problem of approximate parameterized matching was considered under the further restriction that mismatches at any given location could not exceed a predefined maximum number k, and an algorithm was presented working in time $O\left(nk\sqrt{k} + mk \log m\right)$.

Here we focus on the case where both strings are run-length encoded. This case was previously examined in [4] with the further restriction that one of the alphabets is binary. For this special setup, the authors gave a construction working in time $O\Big(n + (r_p \times r_t)\alpha(r_t) \log r_t\Big)$, where r_p and r_t denote the number of runs in the corresponding encodings for p and t, respectively and α is the inverse of Ackerman's function. This complexity actually reduces to $O(n + (r_p \times r_t))$ when both alphabets are binary. (On one hand side it is obvious that the run-length encoding can be computed from the original string in linear time and space while, on the other hand, the original text can be unproportionately long as a function of the run-encoded length. It is also clear that we cannot gain anything without reasonable sized runs, which is equivalent to a relative small number of runs.)

Here we turn our interest to a more general case: we still assume run-length encoded text and pattern, however we relax the constraints on the the size of both alphabets. We give an algorithm, having a time complexity of the form $O\Big((r_t \times r_p) \times F_1 \times F_2\Big)$, where F_1 and F_2 are polynomials of substantial degree in the alphabet size, that reports the text positions where a parameterized match with mismatches between the two run-length encoded strings is achieved within a preassigned bound k.

This paper is organized as follows. In the next section, we give some basic properties, and derive the combinatorial facts used in our construction. Section 3 is devoted to the design and description of our algorithm. The main property subtending to the construction is established in Section 4. The last section lists conclusions and open problems.

2. Problem description

We assume that $\mathbf{x}$ and $\mathbf{y}$ are presented in their run-length encodings. In general that means that the text is given as $\mathbf{x} = x_1^{\alpha_1} x_2^{\alpha_2} \dots x_{r_t}^{\alpha_{r_t}}$ where $x_i \in \Sigma_t$, $x_i \neq x_{i+1}$ and $\sum \alpha_j = n$. Similarly the pattern is $\mathbf{y} = y_1^{\beta_1} \dots y_{r_p}^{\beta_{r_p}}$ (with analogous properties). However here we choose another way describe this encoding method: the text is described as a list of r_t ordered pairs: $\mathbf{x} = \big([L_1, x_1]; [L_2, x_2], \dots, [L_{r_t}, x_{r_t}]\big)$

where $L_1 = 1$ while $L_i = 1 + \sum_{j=1}^{i-1} \alpha_j$. The list $L_1, \ldots, L_{r_t+1}$ is termed to the *left-end* list of the text. This notation is extended to $\mathbf{y}$ in analogy: $\mathbf{y} = \big([\Lambda_1, y_1]; [\Lambda_2, y_2], \ldots, [\Lambda_{r_p}, y_{r_p}]\big)$.

Assume now that we want to compute the approximate parameterized matching of the pattern beginning at location i of $\mathbf{x}$. The substring $\mathbf{x}'$ of length m facing the pattern now is described by ordered pair list $\big([\ell_1, x'_1], [\ell_2, x'_2], \ldots [\ell_k, x'_k]\big)$ where $\ell_1 = 1$, while the list $\ell_2, \ldots, \ell_k$ consists of (in ascending order) those $\ell_j = L - (i-1)$ which satisfy $1 < \ell_j \leq m$ (where L runs the left-end list). The list $\ell_1, \ldots, \ell_k$ is called the *i-current left-end list* and one can imagine it as the corresponding portions of $(i-1)$-left-shifted left-hand list. The letter $x'_1 = \mathbf{x}[i]$ or, with other words, it is $= x_j$ where j is the maximum subscript such that $L_j \leq i$. Furthermore the list $x'_2, \ldots, x'_k$ is equal to the list $x_{j+1}, \ldots x_{j+k-1}$.

Definition 1. *The* i-fusion *(or* fusion *when this causes no ambiguity) is the list $F_i = f_1, \ldots, f_j$ which is the merge of the i-current-left-end list $\ell_1, \ldots, \ell_k$ of the text and the left-end list Λ_1, Λ_{r_p} of the pattern.*

Thus, the elements of the i-fusion F_i can come from the i-current-left-end list of the text, or the left-end list of the pattern, or both. Two elements corresponding to the same aligned position coalesce in a single item and are said to form a *bump*. (In position 1 a bump occurs if and only if position L_j in the left-end list of text is actually equal to i.

Example: To illustrate all these notions assume that the actual portion of the text is $\mathbf{x}[21:42] = 1^1 0^2 1^5 2^2 1^2 0^2 1^3 0^2 1^1 0^3$. With our notation this is

$$\mathbf{x}[21:42] = \big([21,0]; [23,1], [28,2], [30,1], [32,0], [34,1], [37,0], [39,1], [40,0]\big);$$

the elements of the corresponding 22-current-left-end list is $\ell_1 = 1, \ell_2 = 2, \ell_3 = 7, \cdots, \ell_8 = 18, \ell_9 = 19$. (The number of the elements in the current-left-end list may vary the pattern is facing to the text), while the fusion list F_{22} consists of 14 positions ($f_1 = 1, \ldots, f_7 = 13, \ldots f_{14} = 20$). The example in Figure 1 shows all these notations in place:

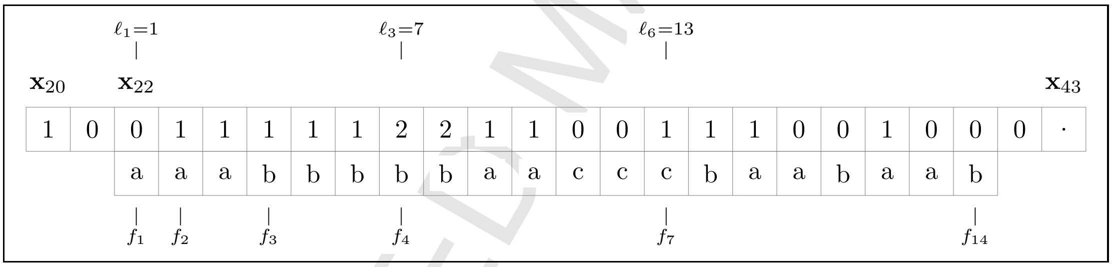

Figure 1: Illustrating the 22-fusion of pattern and text intervals. .

As mentioned, the problem of finding an optimal injection from Σ_p to Σ_t at position i can be re-formulated in terms of the following standard graph theoretic problem.

We are given a weighted bipartite graph G_i with classes Σ_t and Σ_p, which draws its edge-weights from all possible bijections π_i, as follows: for each edge u, v ($u \in \Sigma_p$ and $v \in \Sigma_t$) the weight $w_{u,v}$ is the number of matches induced by accepting $\pi_i(u) = v$.

Under this formulation, an optimal approximate parameterized matching at position i corresponds to a *maximum weighted matching* (*MWM* for short) in a bipartite graph G.. There are several standard methods to determine the best weighted matching in a bipartite graph. However, the complexity of these algorithms is $O\left(V^2 \log V + VE\right)$ (see [8]), which would make the iterated application to our case prohibitive. In what follows, we follow an approach that resorts to MWM more sparsely.

We begin by examining the effect of shifting the text by one position to the left. Clearly, this might change the weight $w_{u,v}$ for every pair. Let $\delta_{u,v}$ be the value of this change, which could be either negative or positive. The new weights after the shift will be in the form $w_{u,v} + \delta_{u,v.}$ Observe that as long as no bump occurs each consecutive shift will cause the same changes in the weights.

Within such a regimen, we could calculate the new weights in our graph following every individual shift, each time at a cost of $O(|\Sigma_t||\Sigma_p|)$ time. But we could as well just use the linear functions $w_{u,v} + \alpha\delta_{u,v}$ to determine the weights of the maximum weighted matching achievable throughout, without computing every intermediate solution.

Whenever a bump occurs, we have to recalculate the δ functions. Each recalculation should take care of all characters that are actually affected by the bump. However, the number of function recalculations cannot exceed $r_t \times r_p$, the maximum number of of bumps.

In conclusion, our task can be subdivided into two interrelated, but computationally distinct, steps:

1. At every bump we have to (re)calculate the function Δ in order to quickly update the weights on the bipartite graph.
2. Within bumps, we have to update the weight function following each unit shift and determine whether or not a change in the matching function is necessary.

3. Parameterized string matching via parametric graph matching

For our intended treatment, we will neglect for a moment the fact that the “weight” and “difference” functions (w and Δ, respectively) take integer values and even that the relative shifts between pattern and text take place in a stepwise discrete fashion.

Definition 2. *Let $G = (A, B, E)$ be a bipartite graph with node sets A and B and edge set E. Assume that $|A| \leq |B|$. A set of independent edges is called* (graph) matching*, and a* full matching *if it covers each vertex in A.*

Let $\mathcal{M}$ denote the set of full matchings. Let $w : E \longrightarrow \mathbb{R}$ and $\Delta : E \longrightarrow \mathbb{R}$ be two given functions on the edges. For some $\lambda \in \mathbb{R}_+$ and for an arbitrary function $z : E \longrightarrow \mathbb{R}$ let $z_\lambda := z + \lambda\Delta$. Furthermore, let

$$L(z) := \max\{z(M) : M \in \mathcal{M}\}$$

and

$$\mathcal{M}_z := \{M \in \mathcal{M} : z(M) = L(z)\}.$$

For the sake of simplicity we set $L(\lambda) := L(w_\lambda)$ and $\mathcal{M}_\lambda := \mathcal{M}_{w_\lambda}$. A fundamental property of the function L is the following

Lemma 1. *$L(\lambda)$ is a convex piecewise linear function.*

Proof: $w_\lambda(M) = w(M) + \lambda\Delta(M)$ is a linear — therefore convex — function of λ for each $M \in \mathcal{M}$. The function $L(\lambda)$ is the maximum of these functions for all $M \in \mathcal{M}$, where $\mathcal{M}$ is a finite set. $\square$

A function $\pi : A \cup B \longrightarrow \mathbb{R}$ is called a *potential* if $\pi(b) \geq 0$ for all $b \in B$. Let as before $z : E \longrightarrow \mathbb{R}$ be an arbitrary weight function on the edges. Then a potential is called *z-feasible* or shortly *feasible* if $z(uv) \leq \pi(u) + \pi(v)$ holds for all $uv \in E$. Finally, let Π_z denote the set of z-feasible potentials. Then, Π_z is a closed convex polyhedron in $\mathbb{R}^{\mathcal{A}\cup\mathcal{B}}$.

The following duality theorem is well known (see e.g. [7]):

Theorem 2.

$$L(z) = \min\left\{\sum_{v\in\mathcal{A}\cup\mathcal{B}} \pi(v) : \pi \in \Pi_z\right\}.$$

If $\pi^* \in \Pi_z$ is an arbitrary minimizing feasible potential, then a full matching M is z-minimal if and only if $z(uv) = \pi^*(u) + \pi^*(v)$ holds for all $uv \in M$. $\square$

From the linearity of the objective function we get the following

Lemma 3. *Let $[\alpha, \beta]$ be a linear segment of $L(\lambda)$. Then $\mathcal{M}_{\lambda_1} = \mathcal{M}_{\lambda_2}$ for all $\lambda_1, \lambda_2 \in (\alpha, \beta)$.* $\square$

Definition 3. *Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be a convex function. A vector $s \in \mathbb{R}^n$ is a* subgradient *of the function f in the point $u \in \mathbb{R}^n$ if $f(v) \geq f(u) + \langle s, v-u \rangle$ holds for all $v \in \mathbb{R}^n$.*

Let $\partial f(u)$ denote the set of the subgradients of f in u, i.e

$$\partial f(u) := \big\{ s \in \mathbb{R}^n : f(v) \geq f(u) + \langle s, v-u \rangle \quad \forall v \in \mathbb{R}^n \big\}. \tag{1}$$

Obviously $\partial f(u)$ is never empty and $|\partial f(u)| = 1$ if and only if f is differentiable in u.

Theorem 4. *For any $\lambda \geq 0$, the value of $L(\lambda)$ and a subgradient of the function L in the point λ can be computed using the max weight matching algorithm.*

Proof: It is easy to see that for any $M \in \mathcal{M}_\lambda$, $\Delta(M)$ is a subgradient of the function L in the point λ. In fact the extremal points of the $\partial L(\lambda)$ can be obtained in this way, i.e.

$$\partial L(\lambda) := \big[\min\{\Delta(M) : M \in \mathcal{M}_\lambda\}, \max\{\Delta(M) : M \in \mathcal{M}_\lambda\} \big].$$

□

Assuming now that a threshold value $\gamma \in \mathbb{R}_+$ is assigned, we look for the set

$$\Gamma := \{\lambda \in \mathbb{R}_+ : L(\lambda) \leq \gamma\}. \tag{2}$$

(When we apply this method for the parameterized string matching problem then $\gamma = k$, but in this proof γ is not necessarily integer.)

Due to the convexity of L, the set Γ is a closed interval. Moreover, it is also easy to see that executing the following Newton-Dinkelbach method from an upper and a lower bounds of Γ gives us the endpoints of Γ in finitely many steps. (See Figure 2 demonstrating the execution of the algorithm.)

```
Procedure Maxl(w,d,lstart)
begin
  l:=lstart;
  do
    M:=max_matching(w+l*d);
    l:=(gamma-w(M))/d(M);
  while (w+l*d)(M)>0;
  return l;
end
```

Using a technique originally developed by Radzik[6], it can be shown that

Theorem 5. *The above method terminates in $O\left(|E|\log^2|E|\right)$ iterations, thus the full running time is $O\left(|B||E|^2\log^2|E| + |B|^3|E|\log^3|E|\right)$.*

We defer the proof of this theorem the next section.

Note that the number of iterations (therefore the running time) is independent from the distance of the initial starting points and from the w and Δ values in the input. It solely depends on the size of the underlying graph.

We now apply the above treatment to our string searching problem. As it has already been mentioned in Section 2, our problem can be considered as a sequence of weighted matching problems over special auxiliary graphs, where an optimal matching in the auxiliary graph represents a best mapping of the pattern alphabet at that position. It has further been noticed that the weights change linearly between two bumps, therefore the problem breaks up into $r_t r_p$ pieces of parametric bipartite graph matching problems (over the integral domain).

First, we mention that restricting ourselves to integer solutions does not cause any problem, as it suffices to round up the solutions into the right direction at the end of the algorithm.

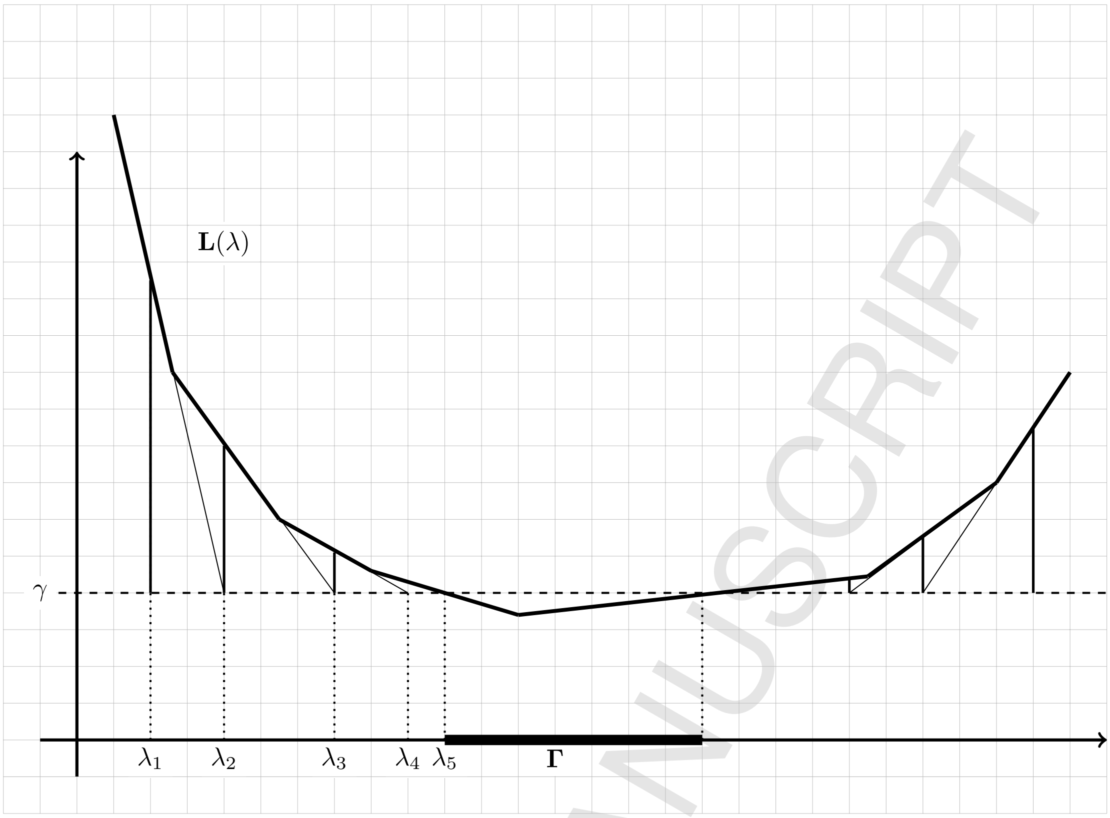

Figure 2: The steps of Newton-Dinkelback method

Now, let us analyze the running time. The nodes of the graph represent the characters of the alphabets, therefore $|A| = |\Sigma_p|$ and $|B| = |\Sigma_t|$, whereas $|E| = |A||B| = |\Sigma_p||\Sigma_t|$. Thus the running time needed to solve a single instance of the parametric weighted matching problem is

$$\begin{aligned} &O\left(|B||A|^2|B|^2\log^2(|A||B|) + |B|^3|A||B|\log^3(|A||B|)\right) \\ &= O\left(|A|^2|B|^3\log^2(|B|) + |B|^4|A|\log^3(|B|)\right) \\ &= O\left(|A||B|^3\log^2|B|(|A| + |B|\log|B|)\right) \\ &= O\left(|A||B|^4\log^3|B|\right) \\ &= O\left(|\Sigma_p||\Sigma_t|^4\log^3|\Sigma_t|\right). \end{aligned}$$

Note that this is simply a constant time algorithm if the size of the alphabets are constant. Thus for any fixed size alphabets the full running time of the algorithm is simply the number of bumps, i.e.

$$O\left(r_p r_t\right). \tag{3}$$

If the size of the alphabet is a parameter, then the full running time is

$$O\left(r_p r_t |\Sigma_p||\Sigma_t|^4\log^3|\Sigma_t|\right). \tag{4}$$

4. Proof of Theorem 5

We prove Theorem 5 by using a technique developed by Radzik [6] to solve the minimum cost-to-time ratio path problem in strongly polynomial time. The proof presented here is an adaptation of the idea to handle matchings instead of paths. Moreover, in our case we must allow negative Δ

components, which also requires special care (and increases the time complexity upper bound by a factor of $\log n$).

Here we examine the case when `lstart` $= 0$ (i.e. when we are looking for the minimum of the interval Γ), the other case is similar. We can assume without loss of generality that $\gamma = 0$. (A possible transformation is to decrease each components of w uniformly by $\gamma/|A|$).

Let $M_1, M_2, \ldots, M_t$ denote the solutions found by the algorithm in the consecutive iterations and let $\lambda_1, \lambda_2, \ldots, \lambda_t$ and $\pi_1, \pi_2, \ldots, \pi_t$ be the corresponding λ values and optimal feasible potentials, respectively.

One can observe that $L(\lambda_1) = w_{\lambda_1}(M_1) > L(\lambda_2) = w_{\lambda_2}(M_2) > \cdots > L(\lambda_t) = w_{\lambda_t}(M_t)$ and $\Delta(M_1) < \Delta(M_2) < \cdots < \Delta(M_t) < 0$ and $\lambda_1 < \lambda_2 < \cdots < \lambda_t$.

A more sophisticated convergence property of the Newton-Dinkelbach method was found by Radzik [6] as follows:

Theorem 6 (Radzik).
$$\frac{L(\lambda_{i+1})\Delta(M_{i+1})}{L(\lambda_i)\Delta(M_i)} \le \frac{1}{4}.$$

□

Definition 4. *Let edge $e \in E$ be called* i-essential *if*

$$e \in M_i \cup M_{i+1} \cup M_{i+2} \cup \cdots .$$

Lemma 7. *Let $k := \left\lceil \frac{\log_2 |E|+3}{2} \right\rceil$. Then for any i at least one of the following holds:*

(a) $\Delta(M_{i+k}) \ge \frac{1}{2}\Delta(M_i)$,

(b) *there exists an i-essential edge e that is not $(i+k)$-essential.*

Proof: Let us assume that (a) does not hold, i.e. $\Delta(M_{i+k}) < \frac{1}{2}\Delta(M_i) < \frac{1}{2}\Delta(M_{i+1}) < 0$. From Theorem 6 we get that
$$L(\lambda_{i+k})\Delta(M_{i+k}) \ge \frac{1}{2|E|} L(\lambda_{i+1})\Delta(M_{i+1}),$$
which yields in turn that
$$L(\lambda_{i+k}) < \frac{1}{|E|} L(\lambda_{i+1}).$$
It is enough to prove that there exist $e \in E$ such that $e \in M_i(e)$ but $e \notin M_j$ for all $j > i+k$.

Let $\tilde{w}_\lambda(uv) := w_\lambda(uv) - \pi_\lambda(u) - \pi_\lambda(v)$. Since π_λ is a feasible potential, $\tilde{w} \le 0$.

$$w_{\lambda_{i+k}}(M_i) = w(M_i) + \lambda_{i+k}\Delta(M_i) \le -L(\lambda_{i+1}) < -|E|L(\lambda_{i+k}),$$

thus

$$\begin{aligned}\tilde{w}_{\lambda_{i+k}}(M_i) = w_{\lambda_{i+k}}(M_i) - \sum_{uv \in M_i} \left(\pi_{\lambda_{i+k}}(u) + \pi_{\lambda_{i+k}}(v)\right) < \\ -|E|L(\lambda_{i+k}) - \sum_{u \in A \cup B} \pi_{\lambda_{i+k}} = -(|E|+1)L(\lambda_{i+k}).\end{aligned}$$

So, there exists $e \in M_i$ such that $\tilde{w}_{\lambda_{i+k}}(e) < -L(\lambda_{i+k})$. Assume that $e \in M_j$. Then

$$\begin{aligned}0 < L(\lambda_j) = w_{\lambda_j}(M_j) \le w_{\lambda_{i+k}}(M_j) = \tilde{w}_{\lambda_{i+k}}(M_j) + \\ + \sum_{uv \in M_j} \left(\pi_{\lambda_{i+k}}(u) + \pi_{\lambda_{i+k}}(v)\right) < -L(\lambda_{i+k}) + L(\lambda_{i+k}) = 0,\end{aligned}$$

therefore we get by contradiction that $e \in M_j$, which completes the proof of Lemma 7. □

Now we can prove Theorem 5 by considering the iterations

$$i = \left\lceil \frac{\log_2 |E| + 3}{2} \right\rceil, 2\left\lceil \frac{\log_2 |E| + 3}{2} \right\rceil, 3\left\lceil \frac{\log_2 |E| + 3}{2} \right\rceil, \ldots$$

and counting how many times the cases (a) and (b) of Lemma 7 may occur.

Case (b) may clearly occur at most $|E|$ times. In order to estimating the number of occurrences of case (a), we use the following theorem of Goemans (published by Radzik in [6]), which states that a geometrically decreasing sequence of numbers constructed in a certain restricted way cannot be exponentially long.

Lemma 8 (Goemans [6]). *Let* $\mathbf{c} = (c_1, c_2, \cdots, c_n)$ *be an* n *dimensional vector with real components, and let* $\mathbf{y_1}, \mathbf{y_2}, \cdots, \mathbf{y_q}$ *be vectors from* $\{-1, 0, 1\}^n$*. If for all* $i = 1, 2, \cdots, q-1$

$$0 < \mathbf{y_{i+1}c} \le \frac{1}{2}\mathbf{y_i c},$$

then $q = O(n \log n)$*.* □

Observe that those $\Delta(M_i)$ values that fall under Case (a) form a sequence of the kind required by the Lemma 8, whence of length $O(|E| \log |E|)$.

5. Conclusion

We have presented a method for computing the parameterized matching on run-length encoded strings over alphabets of arbitrary size. The approach extends to alphabets of arbitrary yet constant size the $O(|r_p| \times |r_t|)$ performance previously available only for binary alphabets. For general alphabets, the bound obtained by the present method exhibits a substantial polynomial dependency on the alphabet size. This, however, should be contrasted with the general version of the problem, that can be solved in time $O(nm(\sqrt{m} + \log n))$. In other words, although the exponents are quite high in our expression, the overall complexity depends – in contrast with the convolution based approaches – on the run-length encoded lengths of the input and it is still polynomial in the size of the alphabets. The problem of designing an alphabet independent $O(|r_p| \times |r_t|)$ time algorithm for this problem is still open.

References

[1] Amir, A., Farach, M., Muthukrishnan, S.: Alphabet Dependence in Parameterized Matching. *Information Processing Letters*, **49**, 111–115 (1994).

[2] Baker, B. S.: Parameterized Duplication in Strings: Algorithms and an Application to Software Maintenance. *SIAM Journal of Computing*, **26** (5) 1343–1362 (1997).

[3] Baker, B. S.: Parameterized Pattern Matching: Algorithms and Applications. *Journal Computer System Science*, **52**, (1) 28–42 (1996).

[4] Apostolico, A. , Erdős, P. L., Lewenstein, M.: Parameterized Matching with Mismatches. *J. Discrete Algorithms* **5** (1), 135–140 (2007).

[5] Hazay,C., Lewenstein, M., Sokol, D.: Approximate Parameterized Matching. *ACM Transactions on Algorithms*, **3** (3) Article 29. (2007).

[6] Radzik, T., Fractional combinatorial optimization, In *Handbook of Combinatorial Optimization.* editors DingZhu Du and Panos Pardalos, vol. 1, Kluwer Academic Publishers, (1998).

[7] Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewood Cliffs, N.J. (1993).

[8] Fredman, M.L., Tarjan, R.E.: Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *Journal of ACM* **34** (3) 596–615 (1987).