Parameterized Searching with Mismatches for Run-Length Encoded Strings
(Extended Abstract)

Alberto Apostolico[1,⋆], Péter L. Erdős[2,⋆⋆], and Alpár Jüttner[3,⋆⋆⋆]

[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
axa@cc.gatech.edu

[2] A. Rényi Institute of Mathematics, Hungarian Academy of Sciences, Budapest, P.O. Box 127, H-1364 Hungary
elp@renyi.hu

[3] Department of Operations Research, Eötvös University of Sciences, Pázmány Péter sétány 1/C, Budapest, H-1117 Hungary
alpar@cs.elte.hu

1 Introduction

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 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].

We study approximate variants of the problem where a (possibly controlled) number of mismatches is allowed. Hence, we are concerned with the second

⋆ 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.

⋆⋆ This work was supported in part by the Hungarian NSF, under contract NK 78439 and K 68262.

⋆⋆⋆ This work was supported by OTKA grant CK80124.

E. Chavez and S. Lonardi (Eds.): SPIRE 2010, LNCS 6393, pp. 365–371, 2010.
© Springer-Verlag Berlin Heidelberg 2010

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 $x_i x_{i+1} \dots x_{i+m-1}$ $(i = 1, 2, \dots n - m + 1)$. 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. An $O\left(nk\sqrt{k} + mk \log m)\right)$ time algorithm for parameterized matching with at most k mismatches was given in [5].

In this paper, we are interested in particular in a more general version where both strings are run-length encoded. This case was previously examined in [4], further restricted to the case where one of the alphabets is binary. For this special case, the authors gave a construction working in time $O\left(n + (r_p \times r_t)\alpha(r_t) \log r_t\right)$, 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.

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 two algorithms, both 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. The first one will compute the parameterized matching with mismatches between two run-length encoded strings giving values throughout the positions of the text; the second will report the positions where such a match is achieved within a preassigned bound k.

2 Problem Description

We assume that $\mathbf{x}$ and $\mathbf{y}$ are presented in their run-length encodings, denoted $X = X_1 X_2 \dots X_{r_t}$ and $Y = Y_1 Y_2 \dots Y_{r_p}$, respectively. The generic run, say X_k corresponds to a maximal substring $x_i x_{i+1} \dots x_{i+\ell-1}$ of consecutive occurrences of the same symbol, and is encoded by the pair $[\sigma, L_k]$ where $\sigma = x_i$, we set x_{n+1} to the empty word, and L_k is the k-th element of the *left-end list*, $L_1 = 1, L_2, \dots, L_{r_t+1} = n + 1$ of $\mathbf{x}$. This notation is extended to Y in analogy.

Consider the left-end list $L_1, \dots, L_{r_t+1}$ of the text and assume that we want to compute the approximate parameterized matching of the pattern beginning at location i of $\mathbf{x}$. It is convenient to view this alignment as a shift of the text $i - 1$ positions to the left. The *i-shift list* is the list $L_1 - (i-1), L_2 - (i-1), \dots, L_{r_t+1} - (i-1)$. At position i, we are interested only in the portion of the text facing the pattern, that is, in the portion of the i-shift list containing the first $|\mathbf{y}| = m$ positive elements.

Definition 1. *The* i-fusion *(or* fusion *when this causes no ambiguity) is the sequence of intervals defined by the left-end list resulting from the merge of the left-end list of* $\mathbf{y}$ *with the i-shift list of* $\mathbf{x}$.

Depending on its origin, an element L_k of a fusion is said to be either a *pattern element* or a *text element*. Two elements of the same value coalesce in a single item and are said to form a *bump*.

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 parametrized 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 show 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 $O\left(|\Sigma_t||\Sigma_p|\right)$ 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 need to 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 matching is* full *if it covers each vertex in A.*

Let $\mathcal{M}$ denote the set of all 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 use the notations $L(\lambda) := L(w_\lambda)$ and $\mathcal{M}_\lambda := \mathcal{M}_{w_\lambda}$. A fundamental (but simple) property of the function L is the following

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

A function $\pi : A \cup B \longrightarrow \mathbb{R}$ is called a *potential* if $\pi(b) \geq 0$ for all $b \in B$. Let $z : E \longrightarrow \mathbb{R}$ be again 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 $\mathbb{R}^{A \cup B}$.

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

Theorem 1

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

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 $uv \in M$.

From the linearity of the objective function we get the following

Claim 2. *Let $[\alpha, \beta]$ be a linear segment of $L(\lambda)$. Then $\mathcal{M}_{\lambda_1} = \mathcal{M}_{\lambda_2}$ for all $\lambda_1, \lambda_2 \in (\alpha, \beta)$.* □

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

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

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

Theorem 2. *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 all the subgradients can be obtained in this way, i.e.

$$\partial L(\lambda) := \{\Delta(M) : M \in \mathcal{M}_\lambda\}.$$

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

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

Due to the convexity of L, the set Γ is a closed interval. Moreover, it is also easy to see that starting 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 1 demonstrating the execution of the algorithm.)

```
Procedure Maxl(w,d,lstart)
begin
  l:=lstart;
  do
    M:=max_matching(w+l*d);
    l:=(w(M)-gamma)/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 3. *The above method terminates in* $O\left(|E|\log^2|E|\right)$ *iterations, thus the full running time is* $O\left(|B||E|^2\log^2|E| + |B|^3|E|\log^3|E|\right)$.

Due to the space limitations the proof of this theorem is deferred to the full paper.

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

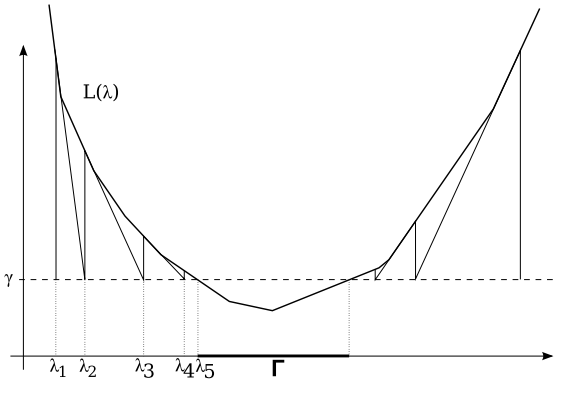

Fig. 1. The steps of Newton-Dinkelback method

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 edge weights change linearly between two bumps, therefore the problem breaks up $r_t r_p$ pieces of parametric bipartite graph matching problems (over the integral domain).

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.

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

$$O\left(|B||A|^2|B|^2\log^2(|A||B|) + |B|^3|A||B|\log^3(|A||B|)\right) = O\left(|\Sigma_p||\Sigma_t|^4\log^3|\Sigma_t|\right).$$

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., $O(r_p r_t)$. If the size of the alphabet is part of the input, then the full running time is $O\left(r_p r_t|\Sigma_p||\Sigma_t|^4\log^3|\Sigma_t|\right)$.

4 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 alphabet of arbitrary yet constant size the $O\left(|r_p| \times |r_t|\right)$ 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: Du, D., Pardalos, P. (eds.) Handbook of Combinatorial Optimization, vol. 1. Kluwer Academic Publishers, Dordrecht (1998)
7. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs (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)