

Parameterized Searching with Mismatches for Run-length Encoded Strings[☆]

Alberto Apostolico¹

*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. Erdős²

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

Alpár Jüttner³

*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 pattern 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)

¹Work 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.

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

³This work was supported by OTKA grant CK80124.

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

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

19 In this paper we study approximate variants of the problem where a (possibly controlled) number
20 of mismatches is allowed. Hence, we are concerned with the second variant. Formally, we seek to find,
21 for given text $\mathbf{x} = x_1x_2 \dots x_n$ and pattern $\mathbf{y} = y_1y_2 \dots y_m$ over respective alphabets Σ_t and Σ_p , the
22 injection π_i from Σ_p to Σ_t maximizing the number of matches between $\pi_i(\mathbf{y})$ and $x_ix_{i+1} \dots x_{i+m-1}$
23 (for all $i = 1, 2, \dots, n - m + 1$).

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

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

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

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

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

53 2. Problem description

54 We assume that \mathbf{x} and \mathbf{y} are presented in their run-length encodings. In general that means that
55 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
56 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 en-
57 coding method: the text is described as a list of r_t ordered pairs: $\mathbf{x} = ([L_1, x_1]; [L_2, x_2], \dots, [L_{r_t}, x_{r_t}])$

94 Within such a regimen, we could calculate the new weights in our graph following every individual
 95 shift, each time at a cost of $O(|\Sigma_t||\Sigma_p|)$ time. But we could as well just use the linear functions
 96 $w_{u,v} + \alpha\delta_{u,v}$ to determine the weights of the maximum weighted matching achievable throughout,
 97 without computing every intermediate solution.

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

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

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

107 3. Parameterized string matching via parametric graph matching

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

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

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

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

117 and

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

118 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
 119 function L is the following

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

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

123 A function $\pi : A \cup B \rightarrow \mathbb{R}$ is called a *potential* if $\pi(b) \geq 0$ for all $b \in B$. Let as before $z : E \rightarrow \mathbb{R}$
 124 be an arbitrary weight function on the edges. Then a potential is called *z -feasible* or shortly *feasible*
 125 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.
 126 Then, Π_z is a closed convex polyhedron in $\mathbb{R}^{A \cup B}$.

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

Theorem 2.

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

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

130 From the linearity of the objective function we get the following

131 **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

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

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

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

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

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

138 **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
 139 λ . In fact the extremal points of the $\partial L(\lambda)$ can be obtained in this way, i.e.

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

140 □

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

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

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

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

```

148 Procedure Maxl(w,d,lstart)
149 begin
150   l:=lstart;
151   do
152     M:=max_matching(w+l*d);
153     l:=(gamma-w(M))/d(M);
154   while (w+l*d)(M)>0;
155   return l;
156 end

```

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

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

160 We defer the proof of this theorem the next section.

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

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

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

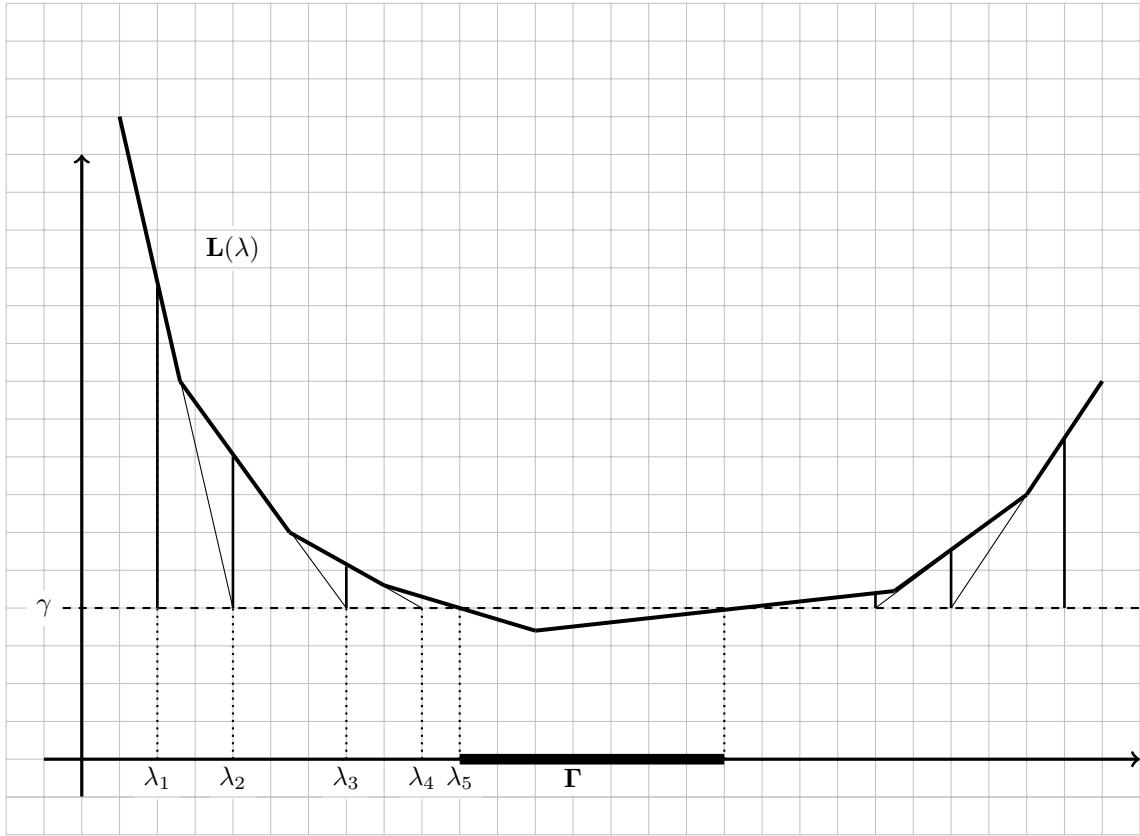


Figure 2: The steps of Newton-Dinkelback method

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

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

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

$$O(r_p r_t). \quad (3)$$

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

$$O(r_p r_t |\Sigma_p||\Sigma_t|^4 \log^3 |\Sigma_t|). \quad (4)$$

176 4. Proof of Theorem 5

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

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

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

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

188 One can observe that $L(\lambda_1) = w_{\lambda_1}(M_1) > L(\lambda_2) = w_{\lambda_2}(M_2) > \dots > L(\lambda_t) = w_{\lambda_t}(M_t)$ and
189 $\Delta(M_1) < \Delta(M_2) < \dots < \Delta(M_t) < 0$ and $\lambda_1 < \lambda_2 < \dots < \lambda_t$.

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

Theorem 6 (Radzik).

$$\frac{L(\lambda_{i+1})\Delta(M_{i+1})}{L(\lambda_i)\Delta(M_i)} \leq \frac{1}{4}.$$

192

□

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

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

193 **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:

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

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

196 **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
197 Theorem 6 we get that

$$L(\lambda_{i+k})\Delta(M_{i+k}) \geq \frac{1}{2|E|}L(\lambda_{i+1})\Delta(M_{i+1}),$$

198 which yields in turn that

$$L(\lambda_{i+k}) < \frac{1}{|E|}L(\lambda_{i+1}).$$

199 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 $j > i+k$.

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

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

thus

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

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

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

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

202 Now we can prove Theorem 5 by considering the iterations

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

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

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

208 **Lemma 8 (Goemans [6]).** *Let $\mathbf{c} = (c_1, c_2, \dots, c_n)$ be an n dimensional vector with real compo-*
 209 *nents, and let $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_q$ be vectors from $\{-1, 0, 1\}^n$. If for all $i = 1, 2, \dots, q - 1$*

$$0 < \mathbf{y}_{i+1} \mathbf{c} \leq \frac{1}{2} \mathbf{y}_i \mathbf{c},$$

210 *then $q = O(n \log n)$.* □

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

213 5. Conclusion

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

224 References

- 225 [1] Amir, A., Farach, M., Muthukrishnan, S.: Alphabet Dependence in Parameterized Matching.
 226 *Information Processing Letters*, **49**, 111–115 (1994).
- 227 [2] Baker, B. S.: Parameterized Duplication in Strings: Algorithms and an Application to Software
 228 Maintenance. *SIAM Journal of Computing*, **26** (5) 1343–1362 (1997).
- 229 [3] Baker, B. S.: Parameterized Pattern Matching: Algorithms and Applications. *Journal Computer*
 230 *System Science*, **52**, (1) 28–42 (1996).
- 231 [4] Apostolico, A., Erdős, P. L., Lewenstein, M.: Parameterized Matching with Mismatches. *J.*
 232 *Discrete Algorithms* **5** (1), 135–140 (2007).
- 233 [5] Hazay, C., Lewenstein, M., Sokol, D.: Approximate Parameterized Matching. *ACM Transactions*
 234 *on Algorithms*, **3** (3) Article 29. (2007).
- 235 [6] Radzik, T., Fractional combinatorial optimization, In *Handbook of Combinatorial Optimization*.
 236 editors DingZhu Du and Panos Pardalos, vol. 1, Kluwer Academic Publishers, (1998).
- 237 [7] Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: *Network Flows: Theory, Algorithms, and Applications*,
 238 Prentice Hall, Englewood Cliffs, N.J. (1993).
- 239 [8] Fredman, M.L., Tarjan, R.E.: Fibonacci Heaps and Their Uses in Improved Network Optimiza-
 240 tion Algorithms. *Journal of ACM* **34** (3) 596–615 (1987).