Constant-space string matching with smaller number of comparisons: sequential sampling

Leszek Gąsieniec† Wojciech Plandowski† Wojciech Rytter†‡

† Instytut Informatki, Uniwersytet Warszawski, Warszawa,
‡ Dept. of Computer Science, University of Bonn, Germany

Abstract

A new string-matching algorithm working in constant space and linear time is presented. It is based on a powerful idea of sampling, originally introduced in parallel computations. The algorithm uses a sample S which consists of two positions inside the pattern P. First the positions of the sample S are tested *against* the corresponding positions of the text T, then a version of Knuth-Morris-Pratt algorithm is applied. This gives the simplest known string-matching algorithm which works in constant space and linear time and which does not use any linear order of the alphabet. A refined version of the algorithm gives the fastest (in the sense of number of comparisons) known algorithm for string-matching in constant space. It makes $(1+\varepsilon)n + O(\frac{n}{m})$ symbol comparisons. This improves substantially the result of [3], where a $(\frac{3}{2}+\varepsilon)n$ comparisons constant space algorithm was designed.

1 Introduction

Assume we are given two strings: a pattern P of length m and a text T of length n. The *string-matching problem* consists in finding all occurrences of P in T.
The algorithms solving this problem with linear cost and (simultaneously) constant space are the most interesting and usually the most sofisticated. By the cost of computations we mean the number of symbol comparisons between pattern and text symbols.

The first algorithm which used constant amount of additional memory was given by Galil and Seiferas in [15]. Later Crochemore and Perrin in [10] have shown how to achieve $2n$ comparisons algorithm preserving small amount of memory.

An alternative algorithm was presented by Gąsieniec, Plandowski and Rytter in [16]. The first small space algorithm which beats the bound of $2n$ comparisons was presented by Breslauer in [3]. He designed $(\frac{3}{2}+\varepsilon)n$ comparisons, constant space algorithm.

In this paper we present constant space and $(1+\varepsilon)n+O(\frac{n}{m})$ comparisons algorithm. For constant space algorithms this bound seems to be tight due to the last results of Breslauer [2]. We introduce also a very simple constant space algorithm with $2n$ comparisons called *sequential sampling*.

All strings considered in the paper are built over a general alphabet Σ (without any restriction). Given a word $w \in \Sigma^*$, by $w[i]$ we mean the symbol at the ith position of the word w. Notice that positions in a word are enumerated from 1.

We say that the word w has *a period per* if and only if $w[i] = w[i+per]$ for all positions $1 \le i \le |w| - per$. The shortest period of w is called *the period* of w. If the period $per \le |w|/2$, then the word w is called *periodic*, otherwise w is *nonperiodic*.

Lemma 1 *If the word w has two periods c, d and $c + d \le |w|$ then w has also a period $gcd(c, d)$, where gcd stands for the greatest common divisor.*

Denote by w^- the word w without its last symbol, then Lemma 1 implies:

Fact 2 *Given periodic word w with the period per, let v be the prefix of w of size per.*
Then the word vv^- is nonperiodic.

The main novel idea behind our algorithm is a use of a two-point sample S of the pattern P. The samples were extensively used in parallel computations, see [19]. Here the sample is used in a sequential setting, another application of sample in sequential computations appears in [8].

Assume that a nonperiodic pattern P has a periodic prefix. Denote by π the longest periodic prefix of P. Let $q-1$ be the length of π, let per be the size of the shortest period of π and let $p = q - per$. Define the set $S = \{ p, q \}$ as the *sample* S of the prefix $P[1..q]$.

Introduce the predicate:

$$MatchSample(i, S) = (T[i+p] = P[p] \text{ and } T[i+q] = P[q])$$

Observation

The positions p, q are the first (from the left) *witnesses against* the period *per* of π.
If $MatchSample(i, S)$ then no occurrence of the pattern starts at any position of T in $[i+2 \ldots i+p]$.

The observation implies that if the pattern matches the text at the positions of the sample, then the next *safe* shift is at least p.

Example

If $P = aaaaaaab$ then we can take $S = \{7, 8\}$. In this case if $MatchSample(i, S)$ then the next shift is at least 7.

Our algorithm is also based on Knuth Morris Pratt (KMP in short) algorithm, see [17].

ALGORITHM KMP;
i:= 0; j:=0;
while $i \leq n - m$ **do**
begin
  $j := \max\{k : T[i+j+1 \ldots i+k] = P[j+1 \ldots k]$ or $k = 0\}$;
  if $j = m$ **then** report match at $i+1$;
  i:= $i + \max(1, j - FT[j])$;
  $j := FT[j]$;
end

The KMP algorithm makes *exact* shifts, using the *failure table* FT, we refer to [12] and [17] for the definition of the failure table. We shall make approximate shifts and use a partial failure table, giving correct values only if they are sufficiently large.

Due to the introduction of the two-point sample the partial failure table will be sufficient.

Such table can be *represented* in constant space. We define later precisely what we mean by a constant space representation of a table.

2 A simple algorithm performing 2n comparisons: sequential sampling

Recall that we call the word w periodic if it has a period $per \leq \frac{1}{2}|w|$. We consider first the case when none of the prefixes of P is periodic. For this case we have a very simple algorithm for text-searching.

ALGORITHM Simple_Text_Searching;
{ none of the prefixes is periodic }
i:= 0;
while $i \leq n - m$ **do**
begin
  $j := \max\{k : T[i+1 \ldots i+k] = P[1 \ldots k]$ *or* $k = 0\}$;
  if $j = m$ **then** report match at $i+1$;
  i:= $i + \lceil \frac{j+1}{2} \rceil$;
end

During every stage (iteration of outer while loop) of the algorithm *Simple_Text_Searching* the total work (number of comparisons) is equal to $j+1$. Since every prefix is nonperiodic we can make shift of size $\lceil \frac{j+1}{2} \rceil$. Notice that $2\lceil \frac{j+1}{2} \rceil \geq j+1$ and we get the algorithm which performs at most $2n$ comparisons.

The second case is when P is nonperiodic and there is a periodic prefix of P.

ALGORITHM SequentialSampling;
{the case when P is nonperiodic and has a sample $S = \{p, q\}$ }
i:= 0;
while $i \leq n - m$ **do**
begin
  { first test positions of the sample {p, q} in P}
  if not $MatchSample(i, S)$ **then** $i := i+1$ **else**
  begin
    $j := \max\{k : T[i+1 \ldots i+k] = P[1 \ldots k]$ *or* $k = 0\}$;
    if $j = m$ **then** report match at $i+1$;
    if $j < q-1$ **then** i:= $i + p$ **else** i:= $i + \lceil \frac{j+1}{2} \rceil$;
  end

Remark We assume that when we compute $j := \max\{k : T[i+1 \ldots i+k] = P[1 \ldots k]$ *or* $k = 0\}$ then the positions $T[i+p]$ and $T[i+q]$ are not tested, since we have already tested them when computing $MatchSample(i, S)$.

In this case the algorithm starts with the test if pattern symbols associated with positions of the sample $\mathcal{S}$ are the same as corresponding symbols in the text. Then we try to match full occurence of prefix $P[1..q]$ without inspecting $P[p]$ and $P[q]$ for the second time. In case a mismatch is found we make shift of size p. Otherwise we start to match longer nonperiodic prefixes and use the approach from the previous algorithm.

We give the name *sequential sampling* to the whole algorithm consisting of subalgorithms for three cases:
(A) the simple case, when the algorithm *Simple_Text_Searching* is applied;
(B) the (main) case of nonperiodic pattern having a sample, when the algorithm *SequentialSampling* is applied;
(C) the case of periodic patterns, which is reduced to one of the cases (A) or (B) in the proof of the theorem below.

Theorem 3 *The algorithm sequential sampling performs at most $2n$ symbol comparisons and uses a constant additional space.*

Proof.
We prove the theorem in three stages:

A: P is nonperiodic and all its prefixes are nonperiodic (the algorithm *Simple_Text_searching* is applied)

The text searching is done as follows inductively. Assume that we start to recognize an occurence of the pattern P at position i in text T. We compare, one by one, pattern and text symbols. Assume that first j comparisons were positive (i.e. $P[1..j] = T[i..i+j-1]$) and $(j+1)$th one was negative ($P[j+1] \neq T[i+j]$). In that case we can make shift of length $s = \lceil \frac{j+1}{2} \rceil$, since prefix $P[1..j]$ is nonperiodic. The work of $j+1$ comparisons is amortized by shift s, since $2 \cdot s \geq j+1$.

B: P is nonperiodic and $P[1..q-1]$ is the longest periodic prefix of P (the algorithm *SequentialSampling* is applied).

Let per be the period of $P[1 \ldots q-1]$. We have that $P[k] = P[k+per]$ for all $1 \leq k \leq q-per-1$. Since prefix $P[1..q]$ is nonperiodic we know that $P[q-per] \neq P[q]$. Let $p = q - per$ and recall that the pair $\mathcal{S} = (P[p], P[q])$ is the sample of P. Negative tests are amortized by immediate shifts, i.e. two comparisons are amortized by shift of length one. In case of positive match of the sample $\mathcal{S}$, we start to test the full match of $P[1..q]$, omitting recognized earlier symbols of the sample. Symbols from $\mathcal{S}$ don't belong to period of the prefix $P[1..q-1]$, so if a mismatch between text and prefix is found we can make shift of length $s = p$. Hence the total work is not greater than $q-1$ and $p \geq \frac{q-1}{2}$ (prefix $P[1..q-1]$ is periodic) so $q \leq 2 \cdot s$ and we get proper amortization. In case the whole prefix $P[1..q]$ was matched all longer prefixes are nonperiodic and we come to case A.

C: Pattern P is periodic with a period per.

Recall that the pattern P is periodic with the period per iff $per \leq \frac{|P|}{2}$, i.e. $P = v^k v'$ where v is the prefix of P of length per, $k \geq 2$ and v' is a prefix of v. We know from fact 2 that the prefix vv^- is nonperiodic. Our algorithm starts with searching for prefix vv^-. One of the cases (A) or (B) is applied. Since the word vv^- is nonperiodic the work to find all its occurrences is amortized by the shifts according to nonperiodic case parts (A and B). If the word vv^- is found then we start to match next symbols of the pattern P. In case of any later mismatch the work equals to $2 \cdot per + k$, for some $0 \leq k \leq m - 2 \cdot per + 1$, but the shift is $s = per + k$, since matched prefix has the period per. Also $2 \cdot s \geq 2 \cdot per + k$ so we get proper amortization in this case, too. Note that in the case the whole pattern is found, the shift equals per and amortize, one to one, comparisons involved in testing first per symbols of the occurrence of P. Since we can remember pattern prefix of size $m - per$, all further comparisons are amortized by one of the shifts explained above.

3 Partial Algorithm KMP

Let FT be the *failure table* of the pattern. The α-*part* of FT is the table:

$$FT_\alpha[j] = \begin{cases} FT[j] & for\ FT[j] > \alpha j \\ 0 & FT[j] \leq \alpha j \end{cases} \quad (1)$$

We say that a table X can be *represented in constant space* if using a precomputed information of a constant size we can compute each value of $X[j]$ in constant time, for any index j.

Lemma 4 *Assume $0 < \alpha,\ \sigma < 1$ are constants. Then the α-part of the failure table corresponding to prefixes longer than σm can be represented in constant space.*

Proof. A technical proof is omitted in this version. ∎

We introduce the function *Partial_KMP* which works in the same way as KMP as long as the needed values of the failure table are avaliable in FT_c.

The main point is that we know that when this function stops the *shift* in the algorithm KMP would be *enough large*.

The partial version of the algorithm KMP is designed as a function. The function starts at the position s of the text assuming the length r prefix of P matches the text and returns the last position of the begining of the pattern with respect to the text. In the first moment the c-part doest not contain the true value of the failure table the algorithm stops.

Obviously the algorithm reports correctly all *matches* that start in the interval $[s..s']$, where s' is the returned value.

Note, that the algorithm uses the table only for prefixes not shorter than cr.

```
function Partial_KMP(c, s, r);
 i:= s − 1; j:=r; { initially T[i + 1 ... i + r] = P[1 ... r]}
 while i ≤ n − m do
 begin j := max{k : T[i + j + 1 ... i + k] = P[j + 1 ... k] or k = 0};
      if j = m then report match at i;
      if j ≤ cr then { the matched prefix is shorter than cr }
           return i and STOP
      if FT_c[j] = 0 then { shift is at least j − cj}
           return max(i, i + j − cj) and STOP
      else begin { FT_c[j] > cj }
           i:= i + ( j − FT_c[j]);
           j := FT_c[j];
           end
 end
return i;
```

Lemma 5 *Assume the function* Partial_KMP(c,s,r) *returns the value* s'. *Then it can be modified to work with* $\frac{n'}{(1-c)} - r + O(\frac{n'}{r})$ *comparisons for* $n' = s' - s$.

Proof. The algorithm makes shifts ($j - FT_c[j]$). Denote by t the number of *matches*. Since at the beginning of the algorithm the length r pattern prefix matches the text the number of matched symbols is $t + r$. It can be $c(t + r)$ matched symbols at the end not amortized by shifts. Thus the number n' of matched symbols amortized by shifts is not less than $(1 - c)(t + r)$, i.e. $t \geq \frac{n'}{(1-c)} - r$. For each mismatch we make a shift. If all shifts are large enough (larger than $r' = \frac{1}{2}(1 - c)r$) then the number of mismatches is $\mathcal{O}(\frac{n'}{r})$. We have to modify slightly the algorithm if the shift is smaller than r'. In this case a prefix of P, of size at least $(1 - c)r$, has a period of size at most $\frac{1}{2}(1 - c)r$. In this moment instead of continuing the algorithm we compute the maximal continuation of such period in the text. This gives us later the shift which is large enough. Then we resume the algorithm given above. We omit the details. ∎

4 Reducing the number of comparisons

In the latter we need to redefine the notion of periodicity. We say that the word w is periodic (or c-periodic) if it has a period $q \leq c|w|$. Observe that each text is 1-periodic.

As previously we search for the pattern P of length m in the text T of length n. The algorithm which is described in this section depends on two constants δ, c such that $\delta > 0$, $0 < c < 1$. Our algorithm makes at most $(1 + \varepsilon(\delta, c))n + \mathcal{O}(\frac{n}{m})$ comparisons and $\lim_{\delta \to 0, c \to 1} \varepsilon(\delta, c) = 0$.

Denote by *OnLine* one of the known online algorithms for string-matching with small number of

comparisons (i.e. $n + \mathcal{O}((n-m)/m)$) and $\mathcal{O}(m)$ space, e.g. the algorithm from [5]. Such algorithm finds at the position i in a text the first from the left occurrence of a pattern making at most $(i+m) + \mathcal{O}(\frac{i}{m})$ comparisons. We shall apply the algorithm *OnLine* to *very short* patterns.

Define another constant M to be the least number K such that the algorithm *OnLine* finds in a given text the first (from the left) occurrence i of length K pattern with at most $(1+\delta)i+$K comparisons. Note, that if δ tends to 0 then M tends to infinity. The pattern (or its prefix) is *very short* iff its length does not exceed M.

Denote by $PREF(k)$ the set of the pattern prefixes longer than k. The behavior of our algorithm depends on the type of the pattern.

Case 1. The pattern is *very short* ($m \leq$M).
Apply the algorithm *OnLine*.

Case 2. One of the prefixes in $PREF($M$)$ is c-periodic with the period shorter than the shortest period of the whole pattern.
Let $pref$ be the longest prefix of P with the period per shorter than $c|pref|$ which is not a period of P. Take the positions $p = |pref| + 1$, $q = p - per$ as the sample S. The algorithm in this case is a modification of the algorithm *SequentialSampling*. We replace the predicate $MatchSample$ by the function $ModifiedMatchSample(i, S)$ which works as follows. If $P[q-1] = P[q]$ then test the position p of the sample first and then the position q, otherwise do it in the reverse order. Return 0 if the sample occurs and the number of made comparisons otherwise. Since $P[q-1] = P[p-1]$ if the first symbol compared by the function matches a symbol in the text, the next shift is at least 2.

```
ALGORITHM Economic_SequentialSampling;
{ P is nonperiodic and p, q give the sample S }
i:= 1;
while i ≤ n − m do
begin k:=ModifiedMatchSample(i, S);
      i:=i + k;
      if k = 0 then begin j:= max{k : P[1..k] = T[i + 1..i + k] or k = 0};
                          if j <= q then i:=i+q
                          else i:=Partial_KMP(1 − c, i, j);
                    end
end
```

Lemma 6 *Assume the pattern satisfies the conditions in Case 2 and $S = \{p, q\}$ is the sample of this pattern. Then the $(1-c)$-part of the failure table for the prefixes in $PREF((1-c)q)$ may be represented in constant space.*

Proof. The prefixes in $PREF(p)$ are either non c-periodic or c-periodic with the period being the shortest period *sh_per* of the pattern. All c-periodic ones are longer than those which are not c-periodic. The values of the $(1-c)$-part of the failure table for non c-periodic prefixes equal 0 and

for length k c-periodic prefixes with the period sh_per equal $k - sh_per$. Hence, having sh_per we are able to retrieve those values in constant time. On the other hand since $p < q + per \leq q + c \cdot p$ we have $p < \frac{q}{(1-c)}$ and, by Lemma 4, the $(1-c)$-part of the failure table for the prefixes longer than $(1-c)q$ and shorter than p can be encoded in constant space. This completes the proof. ∎

Lemma 7 *The algorithm Economic_SequentialSampling makes at most $n/c + O(n/\mathrm{M})$ symbol comparisons and may be implemented in constant space.*

Proof. The fact that the algorithm may be implemented in constant space is a consequence of Lemma 6. Let $shift$ be the shift of the pattern over the text which is made by execution of the instructions inside the while-loop. Let $comp$ be the number of symbol comparisons made during this execution of the instructions. If the sample does not match the text then $shift = comp$. If the sample matches the text and $j \leq q$ then $shift = q \geq comp$ and again the shift amortizes the number of comparisons. In the case $j > q$, the function $Partial_KMP$ is called, and, by Lemma 5, the number of comparisons in this loop is at most $shift/c + O(shift/j)$. Since the sum of all shifts does not exceed n and $j > q > \mathrm{M}$ the result follows. ∎

Case 3: each prefix in $PREF(\mathrm{M})$ is either not c-periodic or its shortest period is the period of the pattern. Denote by $OnLine(i, s)$ the function which starts the algorithm $OnLine$ from the ith position in T and finds the first to the left of i occurrence of length s pattern prefix.

```
ALGORITHM Economic_KMP;
i:= 1;
while i ≤ n − m do
begin i:= OnLine(i,M)
      i:= Partial_KMP(1 − c, i,M));
end
```

The proof of our next lemma is similar to the proof of Lemma 6.

Lemma 8 *Assume the pattern satisfies the requirements of Case 3. Then the $(1-c)$-part of the failure table for the prefixes longer than $(1-c)\mathrm{M}$ may be represented in constant space.*

The upper bound for the number of symbol comparisons in $Economic_KMP$ is a consequence of such upper bounds in algorithms $Online$ and $Partial_KMP$.

Lemma 9 *The algorithm $Economic_KMP$ makes at most $(\max\{1+\delta, 1/c\})n + O(n/\mathrm{M})$ symbol comparisons and may be implemented in constant space.*

The theorem below is a direct consequence of Lemmas 7 and 9 and the fact that if δ tends to 0 then M tends to infinity.

Theorem 10
For any $\varepsilon > 0$ there is a string-matching algorithm working in constant space with $(1+\varepsilon)n + O(\frac{n}{m})$ symbol comparisons.

5 Pattern Preprocessing

Our preprocessing does not assume that the alphabet is linearly ordered, as it is done in [3]. We show the preprocesing only for the case $c = \frac{1}{2}$, the extension to the general case $1 > c > 0$ is possible.

During the preprocessing we compute the period of P (in the case P is periodic) and the longest periodic prefix of P (if there is any). The solution uses iterative approach when we process all pattern prefixes of size 2^i, for all $1 \leq i \leq \lfloor \log m \rfloor$. In case the pattern length is not a power of 2, we run the same algorithm on the pattern extended by some additional dummy symbols. Assume we have

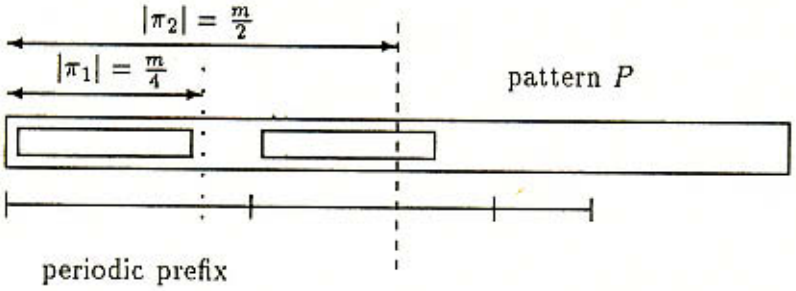

Figure 1: Looking for the longest periodic prefix

already precomputed pattern prefixes π_1, π_2 such that $|\pi_1| = \frac{m}{4}$ and $|\pi_2| = \frac{m}{2}$. To find the longest periodic prefix of the pattern P of length greater then $\frac{m}{2}$ we use the precomputed prefix π_1. Note that the structure of all such prefixes are based on occurences of π_1 in P, see Figure 1. Those occurences could be found by our new searching algorithm *sequential sampling* described before. In case prefix π_1 is nonperiodic it appears in P constant number of times, so all potential periodic prefixes can be checked naively. In case prefix π_1 is periodic with the period per_1, all periodic prefixes can be recognized with support of all proper length sequences of occurences of the word $v_1 v_1^-$ in P, where v_1 is the prefix of P of size per_1. Notice that the period of P corresponds to some periodic prefix. So if we use $T(m)$ to denote the cost of preprocessing of the pattern of length $\frac{m}{2}$ and m we get the inequality $T(m) \leq T(\frac{m}{2}) + O(m)$ which implies $T(m) \leq O(m)$.

References

[1] A.V. Aho, Algorithms for Finding Patterns in Strings. In *Handbook of Theoretical Computer Science*, p. 257–300. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, 1990.

[2] D. Breslauer, private communication.

[3] D. Breslauer, Saving Comparisons in the Crochemore–Perrin String Matching Algorithm. In Proc. of *1st European Symp. on Algorithms*, p. 61–72, 1993.

[4] D. Breslauer and Z. Galil, Efficient Comparison Based String Matching. *J. Complexity* 9(3), p. 339–365, 1993.

[5] R. Cole, R. Hariharan, Tighter bounds on the exact complexity of string matching. In Proc. of *33rd Annual Symp. on Foundations of Comp. Sci.*, p. 600-609, 1992.

[6] R. Cole, R. Hariharan, M.S. Paterson and U. Zwick, Which patterns are hard to find. In Proc. of *2nd Israeli Symp. on Theory of Computing and Systems*, p. 59-68, 1993.

[7] L. Colussi, Correctness and efficiency of string matching algorithms. *Inform. and Control*, 95, p. 225-251, 1991.

[8] M.Crochemore, L. Gąsieniec, W. Plandowski and W. Rytter, Two-dimensional pattern matching in small time and space, accepted to *STACS*'95.

[9] M. Crochemore, String-matching on ordered alphabets. *Theoret. Comput. Sci.*, 92, p. 33-47, 1992.

[10] M. Crochemore and D. Perrin, Two-way string-matching. *J. Assoc. Comput. Mach.*, 38(3), p. 651-675, 1991.

[11] M. Crochemore and W. Rytter, Periodic Prefixes in Texts. In Proc. of *Sequences'91 Workshop "Sequences II: Methods in Communication, Security and Computer Science"*, p. 153-165, Springer-Verlag, 1993.

[12] M. Crochemore and W. Rytter, Text algorithms, *Oxford University Press*, New York, 1994

[13] Z. Galil and R. Giancarlo, On the exact complexity of string matching: lower bounds. *SIAM J. Comput.*, 20(6), p. 1008-1020, 1991.

[14] Z. Galil and R. Giancarlo, The exact complexity of string matching: upper bounds. *SIAM J. Comput.*, 21(3), p. 407-437, 1992.

[15] Z. Galil and J. Seiferas, Time-space-optimal string matching. *J. Comput. System Sci.*, 26, p. 280-294, 1983.

[16] L. Gąsieniec, W. Plandowski and W. Rytter, The zooming method: a recursive approach to time-space efficient string-matching. *Theoret. Comput. Sci.*, to appear.

[17] D.E. Knuth, J.H. Morris and V.R. Pratt, Fast pattern matching in strings. *SIAM J. Comput.*, 6, p. 322-350, 1977.

[18] M. Lothaire, Combinatorics on Words. Addison-Wesley, Reading, MA., U.S.A., 1983.

[19] U.Vishkin, Deterministic sampling - a new technique for fast pattern matching, *SIAM J. Comput.*,20, p. 22-40, 1991.