
Further information on publisher’s website:
https://doi.org/10.1137/17M1120920

Publisher’s copyright statement:
© 2018 SIAM.

Additional information:

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in DRO
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the full DRO policy for further details.
A LINEAR-TIME ALGORITHM FOR MAXIMUM-CARDINALITY MATCHING ON COCOMPARABILITY GRAPHS

GEORGE B. MERTZIOS†, ANDRÉ NICHERLEIN†, AND ROLF NIEDERMEIER†

Abstract. Finding maximum-cardinality matchings in undirected graphs is arguably one of the most central graph problems. For general m-edge and n-vertex graphs, it is well known to be solvable in \(O(m\sqrt{n})\) time. We present a linear-time algorithm to find maximum-cardinality matchings on cocomparability graphs, a prominent subclass of perfect graphs that strictly contains interval graphs as well as permutation graphs. Our greedy algorithm is based on the recently discovered Lexicographic Depth First Search (LDFS).

Key words. greedy algorithms, Lexicographic Depth First Search, perfect graphs, interval graphs

AMS subject classifications. 05C85, 05C70, 05C17

DOI. 10.1137/17M1120920

1. Introduction. The problem MATCHING (or MAXIMUM-CARDINALITY MATCHING) is, given an undirected graph, to compute a maximum-cardinality set of disjoint edges. MATCHING is arguably among the most fundamental graph-algorithmic primitives that can be computed in polynomial time. More specifically, the asymptotically fastest known algorithm for computing a maximum-cardinality matching (subsequently called maximum matching) on an \(n\)-vertex and \(m\)-edge graph runs in \(O(m\sqrt{n})\) time [44]. No faster algorithm is known, even when the given graph is bipartite [24]. Improving this running time, either on general graphs or on bipartite graphs, resisted decades of research. In terms of approximation, it is known that the \(O(m\sqrt{n})\) algorithm of Micali and Vazirani [44] implies a \((1-\epsilon)\)-approximation computable in \(O(m\epsilon^{-1})\) time [13]. For the weighted case, Duan and Pettie [13] provided a linear-time algorithm that computes a \((1-\epsilon)\)-approximate maximum-weight matching (the constant running time factor depending on \(\epsilon\) is \(\epsilon^{-1}\log(\epsilon^{-1})\)). In this work we take a route different from approximation and identify a large graph class, namely cocomparability graphs, on which we show that an optimal solution can be computed in linear time.

Identifying more efficiently solvable special cases for finding maximum matchings has quite some history. Yuster [56] developed an algorithm with running time \(O(rn^2\log n)\), where \(r\) denotes the difference between maximum and minimum vertex degree of the input graph. Moreover, there are (quasi)linear-time algorithms for computing maximum matchings in several special classes of graphs, including interval graphs [30], convex bipartite graphs [51], strongly chordal graphs [10], and chordal bipartite graphs [3]. We refer the reader to Table 1 for a more thorough overview, also including results with superlinear running times. See Figure 1 for an

*Received by the editors March 14, 2017; accepted for publication (in revised form) October 2, 2018; published electronically December 4, 2018.

http://www.siam.org/journals/sidma/32-4/M112092.html

Funding: The first author was partially supported by the EPSRC grant EP/P020372/1. The second author was supported by a postdoc fellowship of the German Academic Exchange Service (DAAD) while at Durham University.

†Department of Computer Science, Durham University, DH1 3LH Durham, UK (george.mertzios@durham.ac.uk).

‡Algorithmics and Computational Complexity, Faculty IV, TU Berlin, 10587 Berlin, Germany (andre.nichterlein@tu-berlin.de, rolf.niedermeier@tu-berlin.de).
Table 1

Fastest algorithms for Matching on special graph classes; \( \omega < 2.373 \) is the matrix multiplication exponent, that is, two \( n \times n \) matrices can be multiplied in \( O(n^\omega) \) time.

<table>
<thead>
<tr>
<th>Graph class</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>( O(m\sqrt{n}) ) [44], ( O(m\sqrt{n}\log(n^2/m)/\log(n)) ) [22], ( O(n^\omega) ) (rand.) [46]</td>
</tr>
<tr>
<td>bipartite</td>
<td>( O(m\sqrt{n}) ) [24], ( O(n^\omega) ) (rand.) [46, 49], ( O(m^{1.43}) ) [33]</td>
</tr>
<tr>
<td>interval</td>
<td>( O(n \log n) ) (given an interval representation) [30, 45]</td>
</tr>
<tr>
<td>circular arcs</td>
<td>( O(n \log n) ) [30]</td>
</tr>
<tr>
<td>cointerval</td>
<td>( O(n \log n + m) ) [19]</td>
</tr>
<tr>
<td>convex</td>
<td>( O(n) ) [51]</td>
</tr>
<tr>
<td>planar</td>
<td>( O(n^{\omega/2}) ) (rand.) [47]</td>
</tr>
<tr>
<td>strongly chordal</td>
<td>( O(n + m) ) (given the strong perfect elimination order) [10]</td>
</tr>
<tr>
<td>chordal bipartite</td>
<td>( O(n + m) ) [3]</td>
</tr>
<tr>
<td>regular</td>
<td>( O(n^2 \log n) ) [56]</td>
</tr>
<tr>
<td>cographs</td>
<td>( O(n) ) (given a cotree) [55]</td>
</tr>
<tr>
<td>cocomparability</td>
<td>( O(n + m) ) (Theorem 2.12 in section 2)</td>
</tr>
</tbody>
</table>

Fig. 1. Overview of most of the graph classes mentioned in Table 1. An edge indicates that the class above strictly contains the class below.

overview concerning the containment relation between the graph classes.

A graph \( G \) is a cocomparability graph if its complement \( \overline{G} \) admits a transitive orientation of its edges. These graphs (as well as their complements, i.e., comparability graphs) arise naturally in several real-world applications as they are closely related to partially ordered sets (also referred to as posets). In particular, a given cocomparability graph \( G \), together with a transitive orientation of the edges of its complement \( \overline{G} \), can be equivalently represented by a poset. Cocomparability graphs have been the subject of intensive theoretical research [4, 6, 7, 11, 14, 15, 26, 27, 29, 39, 43]. On the one hand, cocomparability graphs naturally generalize well-studied graph classes such as interval and permutation graphs [2, 23], trapezoid (or bounded multitolerance) graphs [25, 28, 32, 36, 38, 52], parallelogram (or bounded tolerance) graphs [20, 41, 42], triangle (or \( PP \)) graphs [5, 35], and simple-triangle (or PI) graphs [37, 53, 54]. On the other hand, cocomparability graphs form an “almost maximal” subclass of perfect graphs [2]. Since perfect graphs (as well as comparability graphs) properly contain bipartite graphs (for which improving the \( O(m\sqrt{n}) \) running time is a long-standing open question), it seems out of reach to obtain an algorithm for Matching with linear running time on perfect graphs. Consequently, designing a linear-time algorithm for cocomparability graphs provides a sharp boundary between \( O(n+m) \)-time algorithms and the known \( O(m\sqrt{n}) \)-time algorithms for Matching.

\(^1\)For an overview of the relation between graph classes see http://www.graphclasses.org/.
Our contribution. In this paper we present a linear-time algorithm for MATCHING on cocomparability graphs. It is a simple greedy algorithm, referred to as Rightmost Matching (RMM), running on a specific vertex ordering. Essentially the same greedy approach was earlier considered by Dragan [12] in the context of greedy matchable graphs. The vertex ordering is obtained by using (as a preprocessing step) the recently discovered Lexicographic Depth First Search (LDFS) algorithm [8]. Interestingly it turns out that RMM computes in a trivial way a maximum matching on interval graphs, when applied on the standard interval graph vertex ordering. Note that the class of interval graphs is a strict subset of the class of cocomparability graphs. So far a similar phenomenon of extending an interval graph algorithm to cocomparability graphs by using an LDFS preprocessing step has also been observed for the LONGEST PATH problem [39], the MINIMUM PATH COVER problem [6], and the MAXIMUM INDEPENDENT SET problem [7]. Our results for the RMM algorithm, adding to the previous results [6, 7, 39], provide evidence that cocomparability graphs present an “interval graph structure” when they are considered with an LDFS preprocessing step. This insight is of independent interest and might lead to new and more efficient combinatorial algorithms.

Preliminaries. We use standard notation from graph theory. In particular, all paths we consider are simple paths. A matching in a graph is a set of pairwise disjoint edges. Let $G = (V, E)$ be an undirected graph, and let $M \subseteq E$ be a matching in $G$. A vertex $v \in V$ is called matched with respect to $M$ if there is an edge in $M$ containing $v$; otherwise $v$ is called free with respect to $M$. If the matching $M$ is clear from the context, then we omit “with respect to $M$.” An alternating path with respect to $M$ is a path in $G$ such that every second edge of the path belongs to $M$. An augmenting path is an alternating path whose endpoints are free. It is well known that a matching $M$ is maximum if and only if there is no augmenting path for it [31].

A graph $G = (V, E)$ is an interval graph if we can assign to each vertex of $G$ a closed interval on the real line such that two vertices are adjacent in $G$ if and only if the corresponding two intervals intersect. A comparability graph is a graph whose edges can be transitively oriented; that is, if $u \rightarrow v$ (the edge $\{u, v\}$ is oriented towards $v$) and $v \rightarrow w$, then $u \rightarrow w$. A cocomparability graph $G$ is a graph whose complement $\overline{G}$ is a comparability graph. The class of interval graphs is strictly included in the class of cocomparability graphs [2]. Intuitively, we can transitively orient the “nonedges” of an interval graph, using the following ordering of nonintersecting intervals from left to right: Consider three intervals $I_a, I_b, I_c$ in an interval representation of an interval graph. If $I_a$ lies completely to the left of $I_b$, and $I_b$ lies completely to the left of $I_c$, then also $I_a$ lies completely to the left of $I_c$.

2. A linear-time algorithm for cocomparability graphs. To begin with, we present in subsection 2.1 a simple greedy linear-time algorithm (called RMM) for computing a maximum matching $M$ on interval graphs. Subsequently we provide in subsection 2.2 all necessary background on vertex orderings for cocomparability graphs and on the LDFS, which is needed for our algorithm on cocomparability graphs. Finally, as our central result, we prove in subsection 2.3 that the algorithm RMM actually works also for cocomparability graphs.

---

2Refer to Remark 2 in subsection 2.3 for a discussion about the subtle but important differences from our approach.

3This is the vertex ordering that results from sorting the intervals according to their left endpoints. The RMM algorithm for interval graphs was discovered by Moitra and Johnson [45].

---

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.
2.1. The greedy algorithm for interval graphs. Given an interval graph $G$ with $n$ vertices and $m$ edges, we first compute in $O(n + m)$ time an interval representation of $G$ and, at the same time, we also sort the intervals according to their left endpoint [48]. The algorithm works as follows (cf. [30, 45]):

1. Initialize $M = \emptyset$ and label all vertices as “unvisited.”
2. Pick the unvisited vertex (interval) $x$ which has the rightmost left endpoint among all currently unvisited vertices in $G$. Then, label $x$ as “visited.”
3. If $x$ has at least one unvisited neighbor in $G$, then pick the unvisited neighbor $y$ of $x$ which has the rightmost left endpoint among all unvisited neighbors of $x$. Then, label $y$ as “visited” and add the edge $\{x, y\}$ to $M$.
4. If there is still an unvisited vertex in $G$, then go to step 2.
5. Return $M$.

We call the above algorithm Rightmost Matching (RMM). It can be executed in $O(n + m)$ time; with a simple exchange argument we can show that the matching $M$ returned by RMM is indeed maximum in $G$. This algorithm implicitly uses the following vertex ordering that characterizes interval graphs. It corresponds to sorting the intervals according to their left endpoints and can be computed in $O(n + m)$ time from $G$ [48].

**Lemma 2.1** (see [48]). $G = (V, E)$ is an interval graph if and only if there exists a vertex ordering $\sigma$ of $G$ (called an I-ordering) such that, for all $x <_\sigma y <_\sigma z$, if $\{x, z\} \in E$, then also $\{x, y\} \in E$.

2.2. Cocomparability graphs and vertex orderings. Before we proceed with our algorithm RMM and its analysis on cocomparability graphs (see subsection 2.3), we now state vertex ordering characterizations of cocomparability graphs and of any vertex ordering that can result from an LDFS search on an arbitrary graph. The following vertex ordering characterizes cocomparability graphs [27].

**Definition 2.2** (see [27]). Let $G = (V, E)$ be a graph. An ordering $\pi$ of the vertices $V$ is an umbrella-free ordering (or a CO-ordering) if for all $x <_\pi y <_\pi z$ it holds that if $\{x, z\} \in E$, then $\{x, y\} \in E$ or $\{y, z\} \in E$ (or both).

**Lemma 2.3** (see [27]). A graph $G = (V, E)$ is a cocomparability graph if and only if there exists an umbrella-free ordering $\pi$ of $V$.

Umbrella-free orderings directly generalize I-orderings for interval graphs (see Lemma 2.1). It is worth noting here that, although there exists a linear-time algorithm to compute an umbrella-free ordering $\pi$ of a given cocomparability graph [34], the fastest known algorithm to verify that a given vertex ordering is indeed umbrella-free needs the same time as boolean matrix multiplication (Spinrad [50] discusses this issue). As an example, we illustrate in Figure 2 the cocomparability graph $C_6$, i.e., the complement of the cycle on six vertices. In this graph, it is straightforward to check by Definition 2.2 that the vertex ordering $\pi = (b, d, c, f, e, a)$ is indeed an umbrella-free ordering.

In the following we present the notion of an LDFS ordering $\sigma$ (see Definition 2.5) due to Corneil and Krueger [8]. This notion is based on good triples and bad triples, which are defined next.

**Definition 2.4** (see [8]). Let $G = (V, E)$ be a graph and $\sigma$ be an arbitrary ordering of $V$. Let $a, b, c \in V$ be three vertices such that $a <_\sigma b <_\sigma c$, $\{a, c\} \in E$, and $\{a, b\} \notin E$. If there exists a vertex $d$ such that $a <_\sigma d <_\sigma b$, $\{d, b\} \in E$, and $\{d, c\} \notin E$, then $(a, b, c)$ is a good triple; otherwise it is a bad triple.

**Definition 2.5** (see [8]). Let $G = (V, E)$ be a graph. An ordering $\sigma$ of $V$ is an
Fig. 2. The cocomparability graph \( G = \overline{C_6} \), i.e., the complement of a cycle on six vertices. The vertex ordering \( \pi = (b, d, c, f, e, a) \) is an umbrella-free ordering for \( G \).

Fig. 3. A good triple \((a, b, c)\) and its vertex \( d \) as in Definition 2.4, in the vertex ordering \( \sigma = (a, d, b, c) \). The edges \( \{a, c\} \) and \( \{d, b\} \) are indicated with solid lines and the nonedges \( \{a, b\} \) and \( \{d, c\} \) with dashed lines. Note that \( \{a, d\} \) and \( \{b, c\} \) can be edges or nonedges.

**LDGS ordering if \( \sigma \) has no bad triple.**

An example of a good triple \((a, b, c)\) and the corresponding fourth vertex \( d \) is depicted in Figure 3. Now we present the generic LDFS algorithm (Algorithm 1) due to Corneil and Krueger [8]. LDFS runs on an arbitrary connected graph \( G \), starting at a distinguished vertex \( u \). It is a variation of the well-known Depth First Search (DFS) algorithm; the main difference is that LDFS assigns labels to the vertices and uses the lexicographic order over these labels as a tie-breaking rule. Briefly, it proceeds as follows. Initially, the label \( \varepsilon \) is assigned to every vertex. Then, iteratively, an unvisited vertex \( v \) with a lexicographically maximum label is chosen and removed from the graph. If \( v \) is chosen as the \( i \)th vertex, then the label of each of its unvisited neighbors is being updated by *prepending* the digit \( i \) to it. Note that the digits in the label of any vertex are always in decreasing order. Hence all neighbors of the last chosen vertex have a lexicographically greater label than all its nonneighbors, and thus all vertices are visited in a depth first search order.

**Algorithm 1.** LDFS\((G, u)\) [8].

**Input:** A connected graph \( G = (V, E) \) with \( n \) vertices and a vertex \( u \in V \).

**Output:** An LDFS ordering \( \sigma_u \) of the vertices of \( G \).

1: Assign the label \( \varepsilon \) to all vertices and mark all vertices as unnumbered
2: label\((u) \leftarrow \{0\} \)
3: for \( i = 1 \) to \( n \) do
4:  Pick an unnumbered vertex \( v \) with the lexicographically largest label
5:  \( \sigma_u(i) \leftarrow v \) \{assign to \( v \) the number \( i \); \( v \) is now numbered\}
6:  for each unnumbered vertex \( w \in N(v) \) do
7:     prepend \( i \) to label\((w) \)
8: return the ordering \( \sigma_u = (\sigma_u(1), \sigma_u(2), \ldots, \sigma_u(n)) \)
The execution of the LDFS algorithm is illustrated with the running example of Figure 2. In this example, suppose that the LDFS algorithm starts at vertex $a$. Suppose that LDFS chooses vertex $c$ next. Now, ordinary DFS could choose either $e$ or $f$ next, but LDFS has to choose $e$, since $e$ has a greater label than $f$ ($e$ is a neighbor of the previously visited vertex $a$). The next visited vertex has to be $b$, since it is the only unvisited neighbor of $e$. The vertex following $b$ in the LDFS ordering $\sigma_b$ must be $f$ rather than $d$, since $f$ has a greater label than $d$ ($f$ is a neighbor of vertex $c$ which has been visited more recently than $d$'s neighbor $a$). Finally, LDFS visits the last vertex $d$, completing the LDFS ordering as $\sigma_a = (a, c, e, b, f, d)$.

It is important here to connect the vertex ordering $\sigma_a$ that is returned by the LDFS algorithm (i.e., Algorithm 1) with the notion of an LDFS ordering, as defined in Definition 2.5. The next theorem due to Corneil and Krueger [8] shows that a LDFS algorithm (i.e., Algorithm 1) with the notion of an LDFS ordering, as defined (see Algorithm 1) could choose any of the neighbors $c, d, e$ of the previously visited vertex $a$, since it is the only unvisited neighbor of $e$. The vertex following $b$ in the LDFS ordering $\sigma_b$ must be $f$ rather than $d$, since $f$ has a greater label than $d$ ($f$ is a neighbor of vertex $c$ which has been visited more recently than $d$’s neighbor $a$). Finally, LDFS visits the last vertex $d$, completing the LDFS ordering as $\sigma_a = (a, c, e, b, f, d)$.

It is important here to connect the vertex ordering $\sigma_a$ that is returned by the LDFS algorithm (i.e., Algorithm 1) with the notion of an LDFS ordering, as defined in Definition 2.5. The next theorem due to Corneil and Krueger [8] shows that a LDFS algorithm (i.e., Algorithm 1) with the notion of an LDFS ordering, as defined (see Algorithm 1) could choose any of the neighbors $c, d, e$ of the previously visited vertex $a$, since it is the only unvisited neighbor of $e$. The vertex following $b$ in the LDFS ordering $\sigma_b$ must be $f$ rather than $d$, since $f$ has a greater label than $d$ ($f$ is a neighbor of vertex $c$ which has been visited more recently than $d$’s neighbor $a$). Finally, LDFS visits the last vertex $d$, completing the LDFS ordering as $\sigma_a = (a, c, e, b, f, d)$.

**Algorithm 2. LDFS$^+$ ($G, \pi$).**

**Input:** A connected graph $G = (V, E)$ with $n$ vertices and an ordering $\pi$ of $V$.

**Output:** An LDFS ordering $\sigma$ of the vertices of $G$.

1: Assign the label $\varepsilon$ to all vertices and mark all vertices as unnumbered
2: for $i = 1$ to $n$ do
3: Pick the rightmost vertex $v$ in $\pi$ among the unnumbered vertices with the lexicographically largest label
4: $\sigma(i) \leftarrow v$ \{assign to $v$ the number $i$; $v$ is now numbered\}
5: for each unnumbered vertex $w \in N(v)$ do
6: prepend $i$ to label($w$)
7: return the ordering $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$

Consider our running example of Figure 2. In this graph $G = \overline{C_6}$, suppose that LDFS$^+$ is given as input the umbrella-free ordering $\pi = (b, d, c, f, e, a)$. Then the ordering $\sigma = \text{LDFS}^+(G, \pi)$ is computed using Algorithm 2 as follows. The first visited vertex is $a$, since $a$ is the rightmost vertex in the ordering $\pi$. Now, LDFS (see Algorithm 1) could choose any of the neighbors $c, d, e$ of $a$ next, but LDFS$^+$ (see
Algorithm 2) has to choose \( e \), since \( e \) is the rightmost among these vertices in the ordering \( \pi \). In this example there exists no further tie among vertices with the same label. Thus, proceeding similarly to our example vertex ordering for LDFS above, it follows that the resulting ordering \( \sigma = \text{LDFS}^+(G, \pi) \) is \( \sigma = (a, e, c, f, b, d) \).

For the purposes of our algorithm RMM for computing a maximum matching on cocomparability graphs in subsection 2.3, we will consider an arbitrary umbrella-free vertex ordering \( \pi \) of the input cocomparability graph \( G \), and we will then compute the LDFS ordering \( \sigma = \text{LDFS}^+(G, \pi) \), by applying Algorithm 2 (i.e., LDFS\(^+\)) to \( \pi \). Our RMM algorithm (see Algorithm 3) will then take this LDFS ordering \( \sigma \) as input, together with the graph \( G \). It is important to note here that, starting from an umbrella-free ordering \( \pi \), the LDFS vertex ordering \( \sigma = \text{LDFS}^+(G, \pi) \) remains umbrella-free [6]. That is, \( \sigma \) satisfies both the conditions of Definition 2.2 and Definition 2.5, and thus \( \sigma \) is simultaneously an LDFS ordering and an umbrella-free ordering. For this reason we refer to \( \sigma \) as an LDFS umbrella-free vertex ordering of the input cocomparability graph \( G \). Finally, note that, given an umbrella-free ordering \( \pi \) of a cocomparability graph \( G \) with \( n \) vertices and \( m \) edges, the ordering \( \text{LDFS}^+(G, \pi) \) can be computed in \( O(n + m) \) time [26].

2.3. The algorithm for cocomparability graphs. Once we have computed in \( O(n + m) \) time the LDFS umbrella-free ordering \( \sigma = \text{LDFS}^+(G, \pi) \), we apply our simple linear-time algorithm Rightmost Matching (RMM) (see Algorithm 3) to compute a new vertex ordering \( \widehat{\sigma} \) and a maximum matching \( M \) of \( G \). RMM is a simple greedy algorithm which operates as follows. At every step it visits the rightmost unvisited vertex \( x \) in \( \sigma \) and it labels \( x \) as visited. Then, if \( x \) does not have any unvisited neighbor, then RMM proceeds at the next step by visiting the rightmost currently unvisited vertex in \( \sigma \); note that this vertex is now different from \( x \), as \( x \) has already been labeled as visited. Otherwise, if \( x \) has at least one unvisited neighbor, then RMM visits after \( x \) its rightmost unvisited neighbor \( y \) in the ordering \( \sigma \), and it also adds the edge \( \{x, y\} \) to the computed matching \( M \).

Algorithm 3. RMM\((G, \sigma)\).

**Input:** A cocomparability graph \( G \) with an LDFS umbrella-free ordering \( \sigma \) of \( G \).

**Output:** A vertex ordering \( \widehat{\sigma} \) of \( G \) and a maximum matching of \( G \).

1: Label all vertices “unvisited”; \( i \leftarrow 0 \); \( M \leftarrow \emptyset \)
2: while there are unvisited vertices do
3: \hspace{1em} Pick the rightmost unvisited vertex \( x \) in \( \sigma \) and label \( x \) as “visited”
4: \hspace{2em} \( i \leftarrow i + 1 \); \( \widehat{\sigma}(i) \leftarrow x \) \{add vertex \( x \) to the ordering \( \widehat{\sigma} \)\}
5: \hspace{1em} if \( x \) has at least one unvisited neighbor then
6: \hspace{2em} \hspace{1em} Pick the rightmost unvisited neighbor \( y \) of \( x \) and label \( y \) as “visited”
7: \hspace{2em} \hspace{2em} \( i \leftarrow i + 1 \); \( \widehat{\sigma}(i) \leftarrow y \) \{add vertex \( y \) to the ordering \( \widehat{\sigma} \)\}
8: \hspace{2em} \hspace{2em} \( M \leftarrow M \cup \{x, y\} \) \{match \( x \) and \( y \)\}
9: return the ordering \( \widehat{\sigma} \) and the matching \( M \)

**Remark 1.** Since any I-ordering of an interval graph is also an LDFS umbrella-free ordering (see Lemma 2.1 and Definitions 2.2, 2.4, and 2.5), note that Algorithm 3 also works with an interval graph \( G \) and an I-ordering \( \sigma \) of \( G \) as input. In this case, RMM\((G, \sigma)\) is actually exactly the same RMM algorithm as we sketched in subsection 2.1 for interval graphs.
Remark 2. Essentially the same greedy approach as our RMM algorithm was already considered by Dragan [12]. More specifically, he characterized those graphs \( G \) which admit a vertex ordering \( \tau \) such that the greedy algorithm computes a maximum matching on every induced subgraph \( F \) of \( G \) when applied to the induced subordering of \( \tau \) on the vertices of \( F \). These graphs \( G \) having the above property are called greedy matchable graphs [12]. We prove that cocomparability graphs admit a vertex ordering \( \sigma \) (namely an LDFS umbrella-free ordering) such that the greedy algorithm computes a maximum matching in the input graph \( G \) itself (and not in every induced subgraph of \( G \)). That is, Dragan [12] studied a problem that is very different from computing a maximum matching in a given graph.

Dragan [12] proved that greedy matchable graphs form a subclass of weakly triangulated graphs; a graph is weakly triangulated if it contains neither (as an induced subgraph) a chordless cycle of length at least five nor the complement of such a chordless cycle. On the contrary, cocomparability graphs are not a subclass of weakly triangulated graphs since, for every \( k \geq 3 \), the complement \( \overline{C}_{2k} \) of a chordless cycle with length \( 2k \) is a cocomparability graph. Indeed, the complement of a \( \overline{C}_{2k} \) (i.e., the chordless cycle \( C_{2k} \)) can be transitively oriented. Therefore, our results do not follow from the paper of Dragan [12].

More specifically, one of the main results of Dragan (see Theorem 1 in [12]) is that a graph \( G \) is greedy matchable if and only if \( G \) admits an admissible vertex ordering (as defined in Definition 3 of [12]). Admissible orderings are characterized by the nine forbidden suborderings as shown in Figure 1 of Dragan [12]. However, three of these forbidden suborderings (namely the 2nd, the 5th, and the 9th) are in fact LDFS umbrella-free orderings (see Definitions 2.2 and 2.5 of our paper). To see this, observe that each of these three orderings (i) is umbrella-free (see our Definition 2.2) and (ii) does not contain any triple \( a,b,c \) of vertices such that \( a <_\sigma b <_\sigma c \), \( \{a,c\} \in E \), and \( \{a,b\} \notin E \) (see our Definitions 2.4 and 2.5).

To illustrate this with an example, consider the graph \( G = \overline{C}_6 \) of our Figure 2 and recall from subsection 2.2 that \( \sigma = (a,e,c,f,b,d) \) is an LDFS umbrella-free vertex ordering of \( G \). Note that this ordering \( \sigma \) contains the orderings \( (a,e,c,d) \) and \( (a,e,c,b) \) as induced suborderings. Furthermore, note that these suborderings correspond to the 2nd and the 5th forbidden suborderings of Figure 1 in Dragan’s paper [12], respectively. Hence, \( \sigma \) is an example of an LDFS umbrella-free ordering which is not an admissible ordering; this is an alternative explanation of why our results do not follow from Dragan’s paper [12].

In the remainder of this section, we show that the matching \( M \) returned by \( \text{RMM}(G, \sigma) \) is indeed a maximum matching of \( G \). The proof is by contradiction and uses an appropriate potential function \( f \) that is defined over all matchings of \( G \):

**Definition 2.7 (potential function).** Let \( G = (V,E) \) be a cocomparability graph and \( \sigma \) be an LDFS umbrella-free ordering of \( V = \{v_1, \ldots, v_n\} \) with \( v_1 <_\sigma \cdots <_\sigma v_n \). Let \( M \) be a matching of \( G \). Then the potential function is \( f(M) := \sum_{i=1}^{n} g_M(v_i) \), where for each \( v_i \in V \)

\[
g_M(v_i) := \begin{cases} 
0 & \text{if } \{v_i, v_j\} \in M \text{ and } i < j, \\
(i - j) \cdot (n + 1)^i & \text{if } \{v_i, v_j\} \in M \text{ and } j < i, \\
i \cdot (n + 1)^i & \text{if } v_i \text{ is not matched within } M.
\end{cases}
\]

\( ^4 \)The only difference is that Dragan’s algorithm visits the vertices from left to right and always matches a vertex with its leftmost unvisited neighbor.
Note by Definition 2.7 that, for the empty matching, we have \( f(\emptyset) = \sum_{i=1}^{n} i \cdot (n + 1)^i \). Then, as we add an edge \( \{v_i, v_j\} \) to the current matching \( M \), where \( j < i \) and \( v_i \) and \( v_j \) are unmatched, we have that
\[
\begin{align*}
f(M \cup \{v_i, v_j\}) &= f(M) - i(n + 1)^i - j(n + 1)^j + (i - j)(n + 1)^i \\
&= f(M) - j((n + 1)^j + (n + 1)^i) < f(M).
\end{align*}
\]

Thus, adding edges to a matching decreases the potential function value. The exponential dependency on the vertex-index in \( g_M \) ensures that matching vertices with higher index has a larger impact than matching vertices with lower index. Furthermore, aiming at a small potential function value also means that the endpoints of the matched edges have only a small index difference. We formalize this intuition in the next observation.

Observation 2.8. Let \( G = (V, E) \) be a graph and \( \sigma \) be an arbitrary ordering of \( V = \{v_1, \ldots, v_n\} \) with \( v_1 <_\sigma \cdots <_\sigma v_n \). Let \( M \) and \( M' \) be two different matchings of \( G \) such that at \( v_i \) is the rightmost difference between \( M \) and \( M' \), that is, each vertex \( v_i \), \( \ell > i \), is either free in both \( M \) and \( M' \) or matched with the same \( v_i \) in both \( M \) and \( M' \). Suppose that
- \( \{v_j, v_i\} \in M' \setminus M \), \( j < i \), and
- \( v_i \) is in \( M \) either free or matched to some \( v_j', j' < j \).

Then, \( f(M') < f(M) \).

Proof. We have
\[
f(M') - f(M) = \sum_{k=1}^{n} g_{M'}(v_k) - g_M(v_k) = \sum_{i=k}^{i} g_{M'}(v_k) - g_M(v_k)
\]
as by assumption \( v_i \) is the rightmost vertex where \( M \) and \( M' \) differ. Then,
\[
f(M') - f(M) = g_{M'}(v_i) - g_M(v_i) + \sum_{k=1}^{i-1} g_{M'}(v_k) - g_M(v_k)
\]
\[
< (i - j)(n + 1)^i - (i - x)(n + 1)^i + \sum_{k=1}^{i-1} g_{M'}(v_k),
\]
where \( x = 0 \) if \( v_i \) is free in \( M \) or \( x = j' \) if \( v_i \) is matched to \( v_{j'} \) in \( M \). In both cases we have \( j > x \) and thus
\[
f(M') - f(M) < -(n + 1)^i + \sum_{k=1}^{i-1} g_{M'}(v_k) < -(n + 1)^i + \sum_{k=1}^{i-1} n(n + 1)^k
\]
\[
= -(n + 1)^i + n \frac{(n + 1)^i - 1}{n} - 1
\]
\[
= -(n + 1)^i + (n + 1)^i - 2 < 0.
\]

With the above observation the connection between the RMM algorithm and the potential function \( f \) is easy to see.

Observation 2.9. The matching \( M \) returned by \( RMM(G, \sigma) \) minimizes the function \( f(M) \).
Proof. Let $M'$ be a matching such that $f(M')$ is minimum. Consider the rightmost vertex $v_n$ in the ordering $\sigma$ and let $v_i$ be the rightmost neighbor of $v_n$ in $\sigma$. Assume that $\{v_i, v_n\} \notin M'$. Then let $M''$ be the matching obtained by removing from $M'$ any edges with endpoints $v_i$ or $v_n$, and by adding to it the edge $\{v_i, v_n\}$. By Observation 2.8, we have $f(M'') < f(M')$, a contradiction. Thus $\{v_i, v_n\} \in M'$. We can now recursively apply the same argument in the induced subgraph $G[(V \setminus \{v_i\} \setminus \{v_n\})]$, which eventually implies that $M'$ is the matching returned by RMM($G, \sigma$).

Before we prove our main result in Theorem 2.12, we need to prove a crucial technical lemma (Lemma 2.11). On a high level, our proof strategy is as follows: We consider a maximum matching $M$ minimizing $f$ and a matching $M'$ produced by Algorithm 3. If $M = M'$, then we are done. Otherwise, we take the “rightmost” difference between $M$ and $M'$, that is the rightmost vertex $v$ in $M$ that is not matched in the same way in $M'$ (or $v$ is free in exactly one of the two matchings). Then we show that matching $v$ in $M$ as in $M'$ leads to another maximum matching $M''$ such that $f(M'') < f(M)$. To show this, we make a case distinction where in two cases we need to exclude the special scenario described in Lemma 2.11. The existence of $M''$ would be a contradiction to our choice of $M$. This shows that $M = M'$.

In the next definition we introduce for every vertex $v$ the induced subgraph $G_\sigma(v)$ with respect to the ordering $\sigma$, which is fundamental for the statement and the proof of Lemma 2.11.

**Definition 2.10.** Let $G = (V, E)$ be a cocomparability graph and let $\sigma = (v_1, v_2, \ldots, v_n)$ be an LDFS umbrella-free vertex ordering of $G$. Then, for every $v_i \in V$, the graph $G_\sigma(v_i)$ is the induced subgraph of $G$ on the vertices $\{v_1, v_2, \ldots, v_i\}$.

**Lemma 2.11.** Let $G = (V, E)$ be a cocomparability graph and $\sigma$ be an LDFS umbrella-free ordering of $V$. Let $M$ be a maximum matching of $G$ such that $f(M)$ is minimum among all maximum matchings. Then, there is no quadruple $(a, b, c, x)$ of vertices in $G$ satisfying all of the following six conditions:

1. $a \prec_\sigma b \prec_\sigma c \preceq_\sigma x$,
2. $\{a, c\}, \{b, c\} \in E$ and $\{a, b\} \notin E$,
3. $\{a, c\} \in M$,
4. there is no odd-length alternating path from $a$ to $b$ within $G_\sigma(x)$,
5. there is no odd-length alternating path from $a$ to any free vertex $v$ within $G_\sigma(x)$, and
6. there is no odd-length alternating path from $b$ to any free vertex $v$ within $G_\sigma(x)$.

**Proof.** Let $G$, $\sigma$, and $M$ be as described in the statement of the lemma (see Figure 4). The proof is by contradiction. Towards a contradiction let $(a, b, c, x)$ be a quadruple of vertices satisfying all six conditions of the lemma. Fix now vertex $x$. Among all such quadruples with fixed $x$, let $(a, b, c, x)$ be such that $a$ is leftmost in $\sigma$; that is, for any other such quadruple $(a', b', c', x)$ we have $a \preceq_\sigma a'$. Since $\sigma$ is an LDFS ordering, it follows from conditions 1 and 2 and Definitions 2.4 and 2.5 that there is a vertex $d$ such that $a \prec_\sigma d \prec_\sigma b$, $\{d, b\} \in E$, and $\{d, c\} \notin E$. Since $\{d, c\} \notin E$ and $\sigma$ is umbrella-free, it follows that $\{a, d\} \in E$. Observe that $d$ is matched in $M$ as otherwise conditions 5 and 6 would be violated. Thus, there is a vertex $e \in V$ with $\{e, d\} \in M$. Now we distinguish three cases with respect to the position of $e$ in the ordering $\sigma$.

**Case 1:** $c \prec_\sigma e$. In this case we have that $a \prec_\sigma c \prec_\sigma e$, $\{d, e\} \in E$, and $\{d, c\} \notin E$. Thus, since $\sigma$ is umbrella-free, it follows that $\{c, e\} \in E$. However, in this case for the matching $M' = (M \setminus \{(a, e), \{d, e\}\}) \cup \{(c, e), \{a, d\}\}$ we invoke Observation 2.8.
with $v_i = e$ to obtain $f(M') < f(M)$, a contradiction.

**Case 2:** $a \prec \sigma e \prec \sigma c$. If $(a,e) \in E$, then there exists the length-three alternating path $(a,e,d,b)$ from $a$ to $b$ within $G_{\sigma}(x)$, which is a contradiction to condition 4. Thus, $(a,e) \notin E$. Furthermore, $(c,e) \in E$, since $\sigma$ is umbrella-free and $(a,c) \in E$. Hence, for the matching $M' = (M \setminus \{(a,c),(d,e)\}) \cup \{(e,c),(a,d)\}$ we invoke Observation 2.8 with $v_i = c$ to obtain $f(M') < f(M)$, a contradiction.

**Case 3:** $e \prec \sigma a$. In this case it follows similarly to Case 2 that $(a,e) \notin E$ (proof by contradiction due to condition 4). Furthermore, observe that $(e,d),(a,d) \in E$ and $(e,d) \in M$. Thus the triple $(e,a,d)$ satisfies conditions 1 to 3. Furthermore, if there exists an odd-length alternating path from $e$ to $a$ within $G_{\sigma}(x)$, then this alternating path can be extended through $d$ to an odd-length alternating path from $a$ to $b$ within $G_{\sigma}(x)$, which is a contradiction to condition 4. Hence there is no odd-length alternating path from $e$ to $a$ within $G_{\sigma}(x)$. Similarly, odd-length alternating paths from $e$ (resp., from $a$) to a free vertex $v$ within $G_{\sigma}(x)$ are excluded as well due to condition 5 (resp., due to condition 6). Thus the quadruple $(e,a,d,x)$ satisfies the six conditions of the lemma and it holds that $e \prec \sigma a$, a contradiction to the choice of the initial quadruple $(a,b,c,x)$. \hfill \Box

We are now ready to prove our central result.

**Theorem 2.12.** For any $n$-vertex and $m$-edge cocomparability graph $G$, Algorithm 3 returns a maximum matching $M$ of $G$ in $O(n+m)$ time.

**Proof.** Let $G = (V,E)$ be a cocomparability graph, and let $\sigma$ be an umbrella-free LDFS ordering of $G$. First we prove that Algorithm 3 runs in $O(n+m)$ time. To this end, we denote with $\deg(v)$ the degree of a vertex $v \in V$. During the execution of the algorithm we maintain the unvisited vertices in a doubly linked list $A$ (initially of size $n$), according to their position in $\sigma$. Furthermore, we maintain for each vertex $u$ its unvisited neighbors in a doubly linked list $N_u$ (initially of size $\deg(u)$), again according to their position in $\sigma$. Once we have computed the ordering $\sigma$, the construction of the list $A$ can be done in $O(n)$ time. The construction of all lists $N_u$, where $u \in V$, can be done in $O(n+m)$ time as follows. We initialize $N_u = \emptyset$ for every $u \in V$. Then we iterate for each vertex $u \in V$ in the list $A$ from left to right. For every such vertex $u$ we scan (in an arbitrary order) through its neighborhood $N(u)$ (note that $N_u$ is at this point still incomplete), and for each $v \in N(u)$ we append vertex $u$ in the list $N_v$.

Line 16 can be clearly executed in $O(n)$ time. The rightmost unvisited vertex $x$ in Line 18 can be found in $O(1)$ time as the rightmost vertex in the list $A$. Once $x$ is detected in Line 18, $x$ is removed from $A$ also in $O(1)$ time. Furthermore, $x$ is removed from all lists $N_u$, where $\{x,u\} \in E$, in $O(\deg(x))$ time since $x$ is always
the last element in the respective list. Moreover, Line 19 can be clearly executed in $O(1)$ time. The if-condition of Line 20 can be checked in $O(1)$ time by just checking whether the list $N_x$ is empty. Similarly to Line 18, Line 21 can be executed in $O(\deg(y))$ time. Furthermore, each of Lines 22 to 24 can be clearly executed in $O(1)$ time. Summarizing, the total running time of Algorithm 3 is $O(n + \sum_{u \in V} \deg(u)) = O(n + m)$.

For the correctness part, the proof is done by contradiction. Let $M$ be the matching returned by RMM$(G, \sigma)$. Assume towards a contradiction that $M$ is not a maximum matching. For the rest of the proof, let $M'$ denote a maximum matching that minimizes $f(M')$ among all maximum matchings of $G$. Let $x$ be the rightmost vertex in $\sigma$ on which $M$ differs from $M'$. Then $x$ is matched in at least one of the two matchings $M$ and $M'$. Now we distinguish three cases with respect to the vertex that is matched with $x$ in $M$ and $M'$.

**Case 1:** $x$ is matched in $M'$ to some $y \in V$ but is free in $M$. Then $M$ and $M'$ also differ at vertex $y$. Thus $y <_\sigma x$, since $x$ is the rightmost vertex in which $M$ and $M'$ differ. Consider the iteration $t$ of Algorithm 3 during which the algorithm visits $x$. If $y$ is free in $M$, then this leads to a contradiction; indeed, otherwise Algorithm 3 would have matched $x$ in iteration $t$ as $x$ has at least one unvisited neighbor, namely $y$. Hence, the vertex $y$ is matched in $M$ with a vertex $z$ at an earlier iteration $t' < t$. Then $M$ differs from $M'$ also at vertex $z$. If $z <_\sigma x$, then Algorithm 3 visits $x$ at an earlier iteration than $z$, which is a contradiction to the assumption on $z$. Hence $x <_\sigma z$. This is a contradiction to the assumption that $x$ is the rightmost vertex in $\sigma$ in which $M$ differs from $M'$.

**Case 2:** $x$ is matched in $M$ to some vertex $y \in V$ but is free in $M'$. If $y$ is free in $M'$, then the matching $M' \cup \{(x,y)\}$ is larger than $M'$, which is a contradiction to the maximality assumption on $M'$. Therefore, $y$ is matched in $M'$ to some vertex $z \in V$. Note that $M$ and $M'$ differ also on $y$ and $z$. Thus, it follows by the choice of $x$ that $y <_\sigma x$ and $z <_\sigma x$. Consider now the matching $M'' := (M' \setminus \{(y,z)\}) \cup \{(x,y)\}$, which is maximum since $|M''| = |M'|$. However, invoking Observation 2.8 with $v_i = x$ yields $\sigma f(M'') < f(M')$, which is a contradiction to the assumption on the minimality of $f(M')$.

**Case 3:** $\{x,y\} \in M$ and $\{x,z\} \in M'$ with $z \neq y$. Then $M$ and $M'$ differ also on $y$ and $z$. Thus, it follows by the choice of $x$ that $y <_\sigma x$ and $z <_\sigma x$. Consider the iteration $t$ of Algorithm 3 during which the algorithm visits $x$. Suppose that vertex $z$ is matched in $M$ with a vertex $p$ at an earlier iteration $t' < t$. Then $M$ differs from $M'$ also at vertex $p$. If $p <_\sigma x$, then Algorithm 3 visits $x$ at an earlier iteration than $p$, which is a contradiction to the assumption on $p$. If $x <_\sigma p$, then we have again a contradiction to the assumption that $x$ is the rightmost vertex in $\sigma$ in which $M$ differs from $M'$. Thus $z$ is unmatched in $M$ at the iteration $t$ of Algorithm 3 during which the algorithm visits $x$. Furthermore, $z$ is also unvisited at iteration $t$ since $z <_\sigma x$. Now, if $y <_\sigma z$, then Algorithm 3 would not match $x$ to $y$ at the execution of Line 21, which is a contradiction. Hence $z <_\sigma y$.

Suppose that $y$ is free in $M'$. Then $M'' := (M' \setminus \{(x,z)\}) \cup \{(x,y)\}$ is another maximum matching. Invoking Observation 2.8 with $v_i = x$ yields $f(M'') < f(M')$, a contradiction to the assumption on $M'$. Hence, $y$ is matched in $M'$ to some vertex $w \in V$ with $w <_\sigma x$ by the choice of $x$. If $\{w,z\} \in E$, then the matching $M'':= (M' \setminus \{(x,z),\{w,y\}\}) \cup \{(x,y),\{z,w\}\}$ is another maximum matching. Invoking Observation 2.8 with $v_i = x$ yields $f(M'') < f(M')$, which is a contradiction to the choice of $M'$. Hence $\{z,w\} \notin E$.

Suppose that within $\sigma(x)$ (Definition 2.10) there exists an odd-length alternating
path \( P_0 \) with respect to \( M' \) from \( w \) to \( z \). Let \( E_0 \) be the edges in the path \( P_0 \). Then, swapping in \( M' \) all edges on the path \( P_0 \) (that is, replacing in \( M' \) the edges \( M' \cap E_0 \) with the edges \( E_0 \setminus M' \)), removing \( \{x, z\} \) and \( \{w, y\} \) from \( M' \), and adding \( \{x, y\} \) yields another maximum matching \( M'' \). Recall that \( x \) is the rightmost vertex in which \( M \) and \( M' \) differ. Thus, since the alternating path \( P_0 \) belongs to the induced subgraph \( G_\sigma(x) \), it follows from Observation 2.8 with \( v_1 = x \) that \( f(M'') < f(M') \), a contradiction to the choice of \( M' \). Thus, within \( G_\sigma(x) \) there exists no odd-length alternating path with respect to \( M' \) from \( w \) to \( z \).

Similarly, suppose that within \( G_\sigma(x) \) there exists an odd-length alternating path \( P_1 \) with respect to \( M' \) from \( w \), (resp., from \( z \) to) a free vertex \( v \). Then, swapping in \( M' \) all edges on the path \( P_1 \), removing \( \{x, z\} \) and \( \{w, y\} \) from \( M' \), and adding \( \{x, y\} \) yields another maximum matching \( M'' \) for which Observation 2.8 with \( v_1 = x \) implies \( f(M'') < f(M') \), which is again a contradiction to the choice of \( M' \). Thus there exists within \( G_\sigma(x) \) no odd-length alternating path with respect to \( M' \) from \( w \) (resp., from \( z \) to) a free vertex \( v \).

Now suppose that \( w <_\sigma z \). That is, \( w <_\sigma z <_\sigma y \), where \( \{w, y\} \in E \) and \( \{z, w\} \notin E \). Hence, \( \{z, y\} \in E \) since \( \sigma \) is umbrella-free. Thus, since \( \{w, y\} \notin M' \), it follows that the quadruple \( \{w, z, y, x\} \) satisfies all six conditions in the statement of Lemma 2.11.

This is a contradiction to Lemma 2.11, since \( M' \) is assumed to be a maximum matching of \( G \) such that \( f(M') \) is minimum among all maximum matchings.

Finally, suppose that \( z <_\sigma w \). Recall that \( M \) differs from \( M' \) in \( w \), since \( \{y, w\} \in M' \) and \( \{x, y\} \in M \). Thus, since \( x \) is the rightmost vertex in \( \sigma \) in which \( M \) differs from \( M' \), it follows that \( w <_\sigma x \). That is, \( z <_\sigma w <_\sigma x \), where \( \{x, z\} \in E \) and \( \{z, w\} \notin E \). Hence, \( \{w, x\} \in E \) since \( \sigma \) is umbrella-free. Thus, since \( \{x, z\} \in M' \), it follows that the quadruple \( \{z, w, x, x\} \) satisfies all six conditions in the statement of Lemma 2.11. This is again a contradiction to Lemma 2.11, since \( M' \) is assumed to be a maximum matching of \( G \) such that \( f(M') \) is minimum among all maximum matchings.

Summarizing, the matching \( M \) returned by Algorithm 3 is a maximum matching.

3. Conclusion. We presented a thorough mathematical analysis of an efficient and easy-to-implement linear-time greedy algorithm for computing maximum matchings on cocomparability graphs. This contributes to a long list of polynomial-time algorithms for problems on cocomparability graphs. Notably, most of this previous work showed polynomial-time (typically far from linear) algorithms for problems that are NP-hard on general graphs, while we improved a problem solvable in polynomial time on general graphs to linear time on cocomparability graphs.

Apart from being of interest on its own, our result might also be useful in a more general approach towards deriving faster algorithms for computing maximum matchings in relevant special cases. The fundamental idea behind this, as described in the companion work [40], is as follows. First observe that, once a matching is given which has \( k \) fewer edges than an optimal one, then using \( k \) iterated augmenting path computations (each taking linear time [18]), one can improve it to a maximum matching. If for a graph \( G \) we also have a vertex subset set \( X \), \( |X| = k \), such that \( G - X \) is a cocomparability graph, then for constant \( k \) we could get a linear-time algorithm for MATCHING as follows: First, delete the \( k \) vertices from \( G \), then apply our linear-time algorithm, and then apply (as described above) at most \( k \) iterations of augmenting path computations again with respect to the original graph \( G \), starting with the maximum matching for the cocomparability graph. A drawback of this approach is that we do not even know how to compute in linear time a constant-factor approximation (which would be good enough) for the mentioned vertex deletion set.
of size $k$. Hence, we consider it as an interesting challenge for future work to give a linear-time (constant-factor approximation) algorithm for computing a “minimum-vertex-deletion-to-cocomparability” set. Based on the above considerations, for now we only can state the following result.

**Corollary 3.1.** *Matching can be solved in $O(k \cdot (n + m))$ time when given a size-$k$ vertex set subset $X$ such that deleting $X$ from the given graph yields a cocomparability graph.*

From a more general point of view, Corollary 3.1 is a contribution to the “FPT in P” program [21], heading for more efficient polynomial-time algorithms based on problem parameterizations (also cf. [1, 9, 16, 17]).

**Acknowledgment.** We are deeply grateful to two anonymous reviewers for their very careful reading and their constructive feedback, which helped us significantly clarify and improve the presentation.

**REFERENCES**


