Linear-Time Algorithms for Maximum-Weight Induced Matchings and Minimum Chain Covers in Convex Bipartite Graphs

Boris Klemz* Günter Rote*

November 13, 2017

Abstract

A bipartite graph $G = (U, V, E)$ is *convex* if the vertices in V can be linearly ordered such that for each vertex $u \in U$, the neighbors of u are consecutive in the ordering of V. An *induced matching* H of G is a matching such that no edge of E connects endpoints of two different edges of H.

We show that in a convex bipartite graph with n vertices and m *weighted* edges, an induced matching of maximum total weight can be computed in $O(n + m)$ time.

An *unweighted* convex bipartite graph has a representation of size $O(n)$ that records for each vertex $u \in U$ the first and last neighbor in the ordering of V. Given such a *compact representation*, we compute an induced matching of maximum cardinality in $O(n)$ time.

In convex bipartite graphs, maximum-cardinality induced matchings are dual to minimum *chain covers*. A chain cover is a covering of the edge set by *chain subgraphs*, that is, subgraphs that do not contain induced matchings of more than one edge. Given a compact representation, we compute a representation of a minimum chain cover in $O(n)$ time. If no compact representation is given, the cover can be computed in $O(n + m)$ time.

All of our algorithms achieve optimal running time for the respective problem and model. Previous algorithms considered only the unweighted case, and the best algorithm for computing a maximum-cardinality induced matching or a minimum chain cover in a convex bipartite graph had a running time of $O(n^2)$.

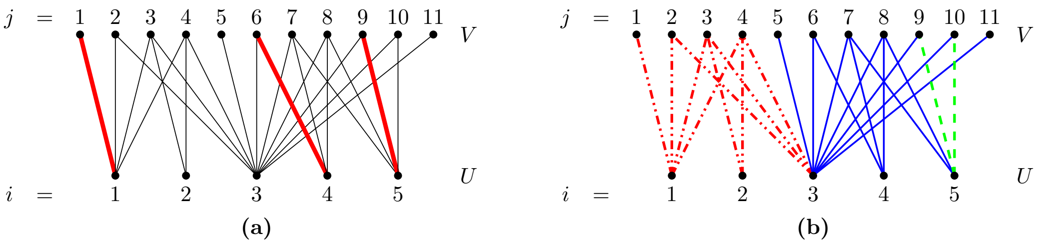
1 Introduction

Problem Statement. A bipartite graph $G = (U, V, E)$ is *convex* if V can be numbered as $\{1, 2, \dots, n_V\}$ so that the neighbors of every vertex $i \in U$ form an *interval* $\{L^i, L^i + 1, \dots, R^i\}$, see Figure 1(a). For such graphs, we consider the problem of computing an induced matching (a) of maximum cardinality or (b) of maximum total weight, for graphs with edge weights.

An *induced matching* $H \subseteq E$ is a matching that results as a subgraph induced by some subset of vertices. This amounts to requiring that no edge of E connects endpoints of two different edges of H, see Figure 1(a). In terms of the line graph, an induced matching is an independent set in the *square* of the line graph. The square of a graph connects every pair of nodes whose distance is one or two. Accordingly, we call two edges of E *independent* if they can appear together in an induced matching, or in other words, if their endpoints induce a $2K_2$ (a disjoint union of two edges) in G. Otherwise, they are called *dependent*.

In convex bipartite graphs, maximum-cardinality induced matchings are dual to minimum chain covers. A *chain graph* Z is a bipartite graph that contains no induced matching of more than one edge, i. e., it contains no pair of independent edges. (Chain graphs are also called *difference graphs* [12] or *non-separable* bipartite graphs [7].) A *chain cover* of a graph G with edge set E is a set of chain subgraphs $Z_1, Z_2, \dots, Z_W$ of G such that the union of the edge sets of $Z_1, Z_2, \dots, Z_W$ is E, see Figure 1(b). A chain cover with W chain subgraphs provides an

*Institut für Informatik, Freie Universität Berlin, Germany



i	L^i	R^i
1	1	4
2	3	4
3	2	11
4	6	8
5	7	10

(c)

Fig. 1: **(a)** A convex bipartite graph $G = (U, V, E)$ containing an induced matching H of size 3. Since we use successive natural numbers as elements of U and V, we will explicitly indicate whether we regard a number x as a vertex of U or of V. There is no induced matching with more than 3 edges: vertex $3 \in U$ is adjacent to all vertices of V except $1 \in V$. Thus, if we match $3 \in U$, this can only lead to induced matchings of size at most 2. Furthermore, we cannot simultaneously match $1 \in U$ and $2 \in U$ since every neighbor of $2 \in U$ is also adjacent to $1 \in U$. **(b)** A minimum chain cover of G with 3 chain subgraphs Z_1, Z_2, Z_3 (in different colors and dash styles), providing an independent proof that H is optimal. Here, Z_1, Z_2, Z_3 have disjoint edge sets, which is not necessarily the case in general. **(c)** The compact representation of G.

obvious certificate that the graph cannot contain an induced matching with more than W edges. We will elaborate on this aspect of a chain cover as a certificate of optimality in Section 5. A *minimum* chain cover of G is a chain cover with a smallest possible number of chain subgraphs. In a convex bipartite graph G, the maximum size of an induced matching is equal to the minimum number of chain subgraphs of a chain cover [24].

We denote the number of vertices by $n_U = |U|$, $n_V = |V|$, $n = n_U + n_V$, and the number of edges by $m = |E|$. If a convex graph is given as an ordinary bipartite graph without the proper numbering of V, it can be transformed into this form in linear time $O(n + m)$ [2]. (In terms of the bipartite adjacency matrix, convexity is the well-known consecutive-ones property.) Unweighted convex bipartite graphs have a natural implicit representation [21] of size $O(n)$, which is often called a *compact representation* [13, 20]: every interval $\{L^i, L^i + 1, \ldots, R^i\}$ is given by its endpoints L^i and R^i, see Figure 1(c). Since the numbering of V can be computed in $O(n + m)$ time, it is easy to obtain a compact representation in total time $O(n + m)$ [20, 22]. The chain covers that we construct will consist of convex bipartite subgraphs with the same ordering of V as the original graph. Thus, we will be able to use the same representation for the chain graphs of a chain cover.

Related Work and Motivation. The problem of finding an induced matching of maximum size was first considered by Stockmeyer and Vazirani [23] as the "risk-free marriage problem" with applications in interference-free network communication. The decision version of the problem is known to be NP-complete in many restricted graph classes [4, 16, 15], in particular bipartite graphs [4, 16] that are C_4-free [16] or have maximum degree 3 [16]. On the other hand, it can be solved in polynomial time in chordal graphs [4], weakly chordal graphs [5], trapezoid graphs, k-interval-dimension graphs and co-comparability graphs [11], amongst others. For a more exhaustive survey we refer to [8].

The class of convex bipartite graphs was introduced by Fred Glover [10], who motivates the computation of matchings in these graphs with industrial manufacturing applications. Items that can be matched when some quantity fits up to a certain tolerance naturally lead to convex bipartite graphs. The computation of matchings in convex bipartite graphs also corresponds to a scheduling problem of tasks of discrete length on a single disjunctive resource [14]. The problem of finding a (classic, not induced) matching of maximum cardinality in convex bipartite graphs has been studied extensively [10, 22, 9] culminating in an $O(n)$ algorithm when a compact representation of the graph is given [22]. Several other combinatorial problems have been studied in convex bipartite graphs. While some problems have been shown to be NP-complete even if restricted to this graph class [1], many problems that are NP-hard in general can be solved efficiently in convex bipartite graphs. For example, a maximum independent set can be found in $O(n)$ time (assuming a compact representation) [20] and the existence of Hamiltonian cycles can be decided in $O(n^2)$ time [18]. For a comprehensive summary we refer to [13].

One of the applications given by Stockmeyer and Vazirani [23] for the induced matching problem can be stated as follows. We want to test (or use) a maximum number of connections between receiver-sender pairs in a network. However, testing a particular connection produces noise so that no other node in reach may be tested simultaneously. We remark that this type of motivation extends very naturally to convex bipartite graphs when we consider *wireless* networks in which nodes broadcast or receive messages in specific frequency ranges. Further, *weighted* edges can model the importance of connections.

Previous Work. Yu, Chen and Ma [24] describe an algorithm that finds both a maximum-cardinality induced matching and a minimum chain cover in a convex bipartite graph in runtime $O(m^2)$. Their procedure is improved by Brandstädt, Eschen and Sritharan [3], resulting in a runtime of $O(n^2)$. Chang [6] computes maximum-cardinality induced matchings and minimum chain covers in $O(n+m)$ time in bipartite permutation graphs, which form a proper subclass of convex bipartite graphs. Recently, Pandey, Panda, Dane and Kashyap [19] gave polynomial algorithms for finding a maximum-cardinality induced matching in circular-convex and triad-convex bipartite graphs. These graph classes generalize convex bipartite graphs.

Our Contribution. We improve the previous best $O(n^2)$ algorithm [3] for maximum-cardinality induced matching and minimum chain covers in convex bipartite graphs in several ways. In Section 2 we give an algorithm for finding maximum-weight induced matchings in convex bipartite graphs with $O(n+m)$ runtime. The weighted problem has not been considered before. In Section 3 we specialize our algorithm to find induced matchings of maximum cardinality in $O(n)$ runtime, given a compact representation of the graph. In Section 4 we extend this approach to obtain in $O(n)$ time a compact representation of a minimum chain cover. If no compact representation is given, our approach is easily adapted to produce a minimum chain cover in $O(n+m)$ time.

All of our algorithms achieve optimal running time for the respective problem and model. Our results for finding a maximum-cardinality induced matching also improve the running times of the algorithms of Pandey et al. [19] for the circular-convex and triad-convex case, as they use the convex case as a building block.

2 Maximum-Weight Induced Matchings

In this section, we compute a maximum-weight induced matching of a given edge-weighted convex bipartite graph $G = (U, V, E)$ in time $O(n + m)$. We generally write indices $i \in U$ as superscripts and indices $j \in V$ as subscripts. We consider E as a subset of $U \times V$. We assume that $V = \{1, \ldots, n_U\}$ is numbered as described in Section 1 and the interval $\{L^i, L^i{+}1, \ldots, R^i\} \subseteq V$ of each vertex $i \in U$ is given by the pair (L^i, R^i) of the left and right endpoint. Each edge $(i, j) \in E$ has a weight C^i_j.

Our dynamic-programming approach considers the following subproblems: For an edge $(i, j) \in E$, we define W^i_j as the cost of the maximum-weight induced matching that uses the edge (i, j) and contains only edges in $U \times \{1, \ldots, j\}$. The following dynamic-programming recursion computes W^i_j:

$$W^i_j = C^i_j + \max\{\, W^{i'}_{j'} \mid R^{i'} < j,\ j' < L^i \,\} \cup \{0\} \tag{1}$$

The range over which the maximum is taken is illustrated in Figure 2. In this recursion, we build

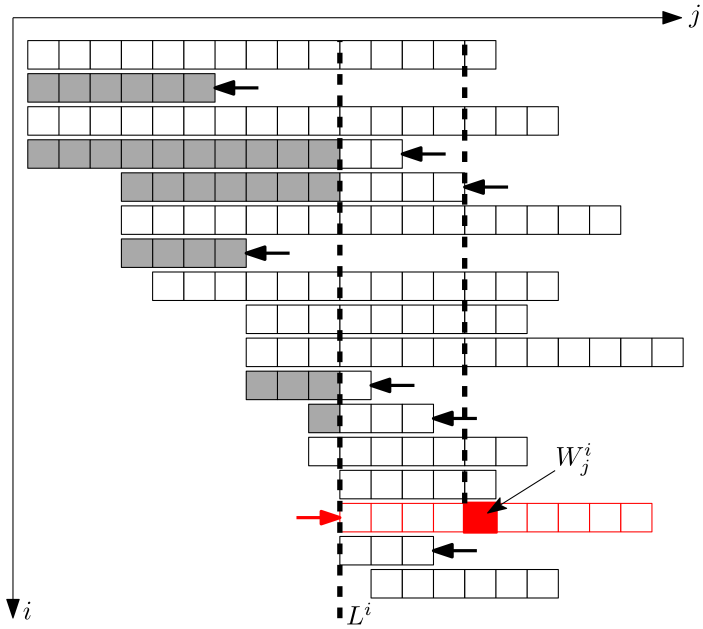


Fig. 2: The table entries that go into the computation of W_j^i are shaded: They lie in rows that end to the left of W_j^i (marked by an arrow), and only the entries to the left of L^i are considered.

the induced matching H of weight W_j^i by adding the edge (i, j) to some induced matching H' of weight $W_{j'}^{i'}$. We want H to be an induced matching: By construction, the edge (i', j') is independent of (i, j), but we have to show that the other edges of H' are also independent of (i, j). In order to prove this (Lemma 2), we use a transitivity relation between independent edge pairs.

Observation 1. *Two edges (i, j) and (i', j') are independent if and only if $j' \notin [L^i, R^i]$ and $j \notin [L^{i'}, R^{i'}]$.*

Lemma 1. *Let $(i'', j''), (i', j'), (i, j) \in E$ with $j'' < j' < j$. Assume that (i'', j'') and (i', j') are independent, and (i', j') and (i, j) are independent. Then (i'', j'') and (i, j) are independent.*

Proof. By Observation 1, we have $j'' \le R^{i''} < j' \le R^{i'} < j$ and $j'' < L^{i'} \le j' < L^i \le j$. Thus, $j \notin [L^{i''}, R^{i''}]$ and $j'' \notin [L^i, R^i]$. □

Lemma 2. *The recursion* (1) *is correct.*

Proof. By Observation 1, any edge (i', j') with $j' < j$ that is independent of (i, j) satisfies $R^{i'} < j$ and $j' < L^i$. By Lemma 1, all other edges (i'', j'') used to obtain the matching value $W_{j'}^{i'}$ are also independent of (i, j). □

We create a table in which we record the entries W_j^i. We assume that the intervals are sorted in nondecreasing order by L^i, that is, $L^i \le L^h$ for $i < h$. The values $W_{L^i}^i, \dots, W_{R^i}^i$ form the i-th *row* of the table. We fill the table row by row proceeding from $i = 1$ to $i = n_U$. Each row i is processed from left to right.

The only challenge in evaluating (1) is the maximum-expression, for which we introduce the notation M_j^i.

$$M_j^i = \max\{\, W_{j'}^{i'} \mid R^{i'} < j,\ j' < L^i \,\} \cup \{0\}$$

We discuss the computation of the leftmost entry $W_{L^i}^i$ later. When we proceed from W_j^i to W_{j+1}^i we want to go incrementally from M_j^i to M_{j+1}^i. Direct comparison of the respective defining sets leads to

$$M_{j+1}^i = \max\ \{M_j^i\} \cup \{\, W_{j'}^{i'} \mid R^{i'} = j,\ j' < L^i \,\} \tag{2}$$

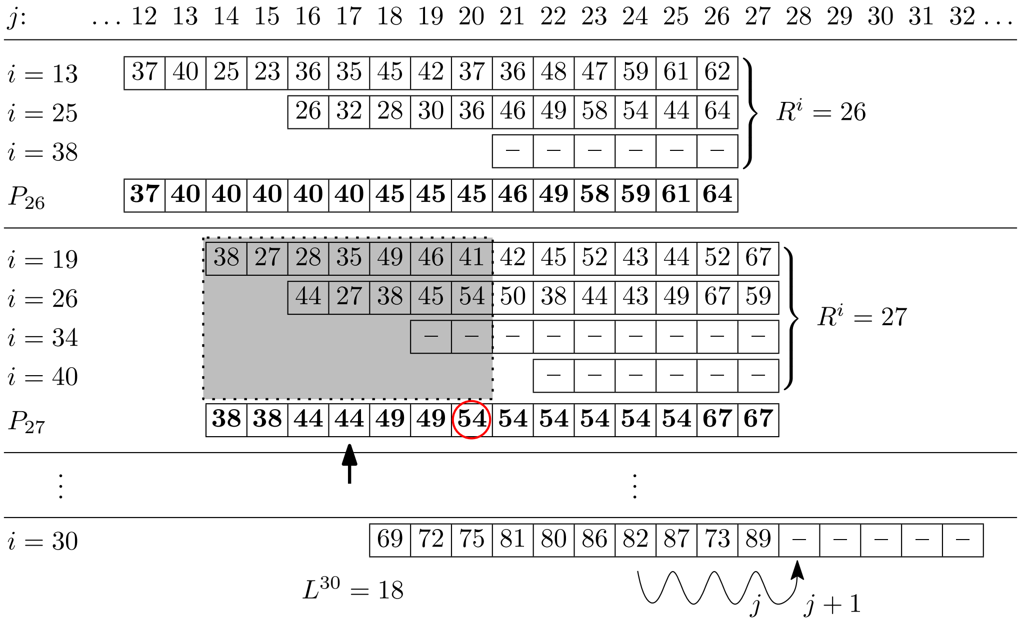


Fig. 3: Example. We are in the process of filling row 30 from left to right. All rows with smaller index i have been processed and are filled with the entries W^i_j. Unprocessed entries are marked as "–". The figure does not show the rows in the order in which they are processed, but intervals with the same right endpoint $R^i = r$ are grouped together. The bold entries collect the provisional maxima P_r in each group. By way of example, the encircled entry $P_{27}[20] = 54$ is the maximum among the shaded entries of the intervals that end at $R^i = 27$, ignoring the yet unprocessed entries. As we proceed from $j = 27$ to $j = 28$ in row 30, the intervals with $R^i = 27$ become relevant. The maximum usable entry from these intervals is found in position 17 of this array, because $17 = L^{30} - 1$. The entry $P_{27}[17] = 44$ is marked by an arrow. The next entry W^{30}_{28} will be computed as $C^{30}_{28} + \max\{P_{27}[17], P_{26}[17], \dots, P_{17}[17]\}$. (Some of these entries might not exist.) We can observe that the minimum over which $P_{27}[17]$ is defined involves no unprocessed entries (Lemma 3). When the next row $i = 34$ in the group with $R^i = 27$ is later filled, it will be necessary to update P_{27}.

In order to evaluate the maximum of the second set in (2) efficiently, we group intervals i' with a common right endpoint $R^{i'} = r$ together. Let S_r be the earliest startpoint of an interval with endpoint r. If there are no intervals with endpoint r, we set $S_r := r$. (It would be more logical to set $S_r := r + 1$ in this case, but this choice makes the algorithm simpler.) We maintain an array $P_r[j]$ for $S_r \le j \le r$ that is defined as follows:

$$\begin{aligned} P_r[j] := \max\{\, W^{i'}_{j'} \mid R^{i'} &= r, \\ &\text{row } i' \text{ has already been processed}, \\ &j' \le j \,\} \cup \{0\} \end{aligned}$$

In a sense, $P_r[j]$ is a provisional version of the expression $\max\{\, W^{i'}_{j'} \mid R^{i'} = r,\ j' < j \,\}$, which takes into account only the already processed rows. For (2), we need the entry $P_j[L^i - 1]$, and we will see that all relevant entries have already been computed whenever we access this entry. Thus, we rewrite (2):

$$M^i_{j+1} = \begin{cases} \max\{M^i_j, P_j[L^i - 1]\}, & \text{if } L^i - 1 \ge S_j \text{ and, thus, } P_j[L^i - 1] \text{ is defined} \\ M^i_j, & \text{otherwise} \end{cases} \tag{3}$$

The condition $L^i - 1 \ge S_j$ ensures that the array index $L^i - 1$ does not exceed the left boundary of the array P_j. Also, the index $L^i - 1$ never exceeds the right boundary j of the array P_j, since $L^i < j+1 \le R^i$, and therefore $L^i - 1 \le j$. Thus, $P_j[L^i - 1]$ is always defined when it is accessed.

Lemma 3. *When entry W^i_{j+1} is processed,* (2) *and* (3) *define the same quantity M^i_{j+1}.*

Proof. We distinguish three cases.

Case 1: No interval ends at j, and accordingly, $S_j = j$.

In this case $M^i_{j+1} = M^i_j$ in (2) since its rightmost set is empty. Since $L^i < j+1 \le R^i$, we have $L^i - 1 < S_j = j$ and, thus, the right side of (3) evaluates also to M^i_j.

Case 2: There exists an interval ending at j, and $L^i - 1 < S_j$. The right side of (3) evaluates to M^i_j. In (2), intervals i' that end at $R^{i'} = j$ have $L^{i'} \ge S_j > L^i - 1$. Thus, an edge (i', j') with $j' < L^i$ and $R^{i'} = j$ does not exist, and the second set in (2) is empty. Therefore, (2) evaluates to $M^i_{j+1} = M^i_j$.

Case 3: There exists an interval ending at j, and $L^i - 1 \ge S_j$. In this case, $P_j[L^i - 1]$ is defined:

$$P_j[L^i - 1] = \max\{\, W^{i'}_{j'} \mid R^{i'} = j,\ j' \le L^i - 1,\ \text{row } i' \text{ already processed} \,\} \tag{4}$$

For each entry $W^{i'}_{j'}$ with $j' < L^i$, we conclude that $L^{i'} \le j' < L^i$ and, thus, row i' has already been processed. This means that the condition that row i' was processed is redundant, and (4) coincides with the right side of (2). □

After processing row i with startpoint $\ell = L^i$ and endpoint $r = R^i$, we have to update the values in $P_r[j]$. This is straightforward. Figure 3 illustrates the role of the arrays $P_r[j]$ when processing a row.

It remains to discuss the computation of the first value W^i_ℓ of the row. An edge $(i', j'), j' < \ell$ and edge (i, ℓ) are independent if and only if the interval i' ends before ℓ, that is $R^{i'} < \ell$. Since we process the intervals in nondecreasing order by their startpoints, it suffices to maintain a value F with the maximum $W^{i'}_{j'}$ in all *finished* intervals: those intervals i' that end before ℓ. In other words $F = \max\{P_1[1], P_2[2], \dots, P_{\ell-1}[\ell - 1]\}$. This value is easily maintained by updating F as ℓ increases. The full details are stated as Algorithm 1.

The update of the array $P_r[j]$ in the second loop can be integrated with the computation of W^i_j in the first loop. When this is done, the values W^i_j need not be stored at all because they are not used. As stated earlier, when no interval ends at a point $r \in V$, we set $S_r = r$. The array P_r consists of a single dummy entry $P_r[r] = 0$. This way we avoid having to treat this special cases during the algorithm.

We have described the computation of the *value* of the optimal matching. It is straightforward to augment the program so that the optimal matching itself can be recovered by backtracking how the optimal value was obtained, but this would clutter the program.

Theorem 1. *A maximum-weight induced matching of an edge-weighted convex bipartite graph can be computed in $O(n+m)$ time.*

3 Maximum-Cardinality Induced Matchings

For the unweighted version of the problem, we assume a compact representation of a convex bipartite graph $G = (U, V, E)$, that is, for each $i \in U$ we are given the startpoint L^i and endpoint R^i of its interval $\{L^i, L^i + 1, \dots, R^i\}$. This makes it possible to obtain a linear runtime of $O(n)$.

The recursion (1) can be specialized to the unweighted case by setting $C^i_j \equiv 1$.

$$W^i_j = 1 + \max\{\, W^{i'}_{j'} \mid R^{i'} < j,\ j' < L^i \,\} \cup \{0\} \tag{5}$$

Algorithm 1: Weighted Maximum Matching

$\triangleright$ Preprocessing:
for $r := 1$ **to** n_V **do**
    Find startpoint S_r of the longest interval $[S_r, r]$ with endpoint r
    Create an array $P_r[S_r \,..\, r]$ and initialize it to 0.
    (If there is no such interval with endpoint r, set $S_r := r$ and create an array with a single dummy entry $P_r[r]$ that will remain at 0.)
$\triangleright$ Main program:
$F := 0$ $\triangleright$ maximum entry in finished intervals
for $\ell := 1$ **to** n_U **do**
    $\triangleright$ $F = \max\{P_1[1], P_2[2], \dots, P_{\ell-1}[\ell-1]\}$
    for all *rows* $i \in U$ *with* $L^i = \ell$ **do** $\triangleright$ Process each interval i that starts at ℓ
        $r := R^i$
        $\triangleright$ Process the i-th interval $[L^i, R^i] = [\ell, r]$ and fill row i of the table:
        $M := M^i_\ell := F$ $\triangleright$ M will be the current value of M^i_j
        $W^i_\ell := C^i_\ell + M$ $\triangleright$ leftmost entry
        for $j := \ell + 1$ **to** r **do** $\triangleright$ compute successive entries
            if $S_j \le \ell - 1$ **then**
                $M := \max\{M, P_{j-1}[\ell-1]\}$ $\triangleright$ $M^i_j := \max\{M^i_{j-1}, P_{j-1}[\ell-1]\}$
            $W^i_j := C^i_j + M$
        $\triangleright$ Go through the computed entries again to update the array P_r:
        $q := 0$ $\triangleright$ the row maximum so far
        for $j := \ell$ **to** r **do**
            $q := \max\{q, W^i_j\}$ $\triangleright$ $q = \max\{0, W^i_\ell, W^i_{\ell+1}, \dots, W^i_j\}$
            $P_r[j] := \max\{P_r[j], q\}$
    $F := \max\{F, P_\ell[\ell]\}$ $\triangleright$ update F as ℓ is incremented
return F $\triangleright$ the maximum weight of an induced matching

This recursion has already been stated in [24] and [3] in a slightly different formulation. Yu, Chen and Ma [24] describe it as a greedy-like procedure that "colors" the edges of a bipartite graph with the values W^i_j. From this coloring, they obtain both a maximum-cardinality induced matching and a minimum chain cover. The original implementation given in [24] runs in time $O(m^2)$. Brandstädt, Eschen and Sritharan [3] give an improved implementation of the coloring procedure with runtime $O(n^2)$. Our Algorithm 1 from Section 2 obtains the values W^i_j in total time $O(n + m)$.

Given a compact representation, we can exploit some structural properties of the filled dynamic-programming table to further improve the runtime to $O(n)$. The following observations were first given in [24] and [3].

Lemma 4 ([24, Lemma 5])**.** *The values W^i_j are nondecreasing in each row.*

Proof. This is obvious from (5), since the set over which the maximum is taken increases with j. □

Lemma 5 ([3, Lemma 3.3, Lemma 3.4])**.** *Each row contains at most two consecutive values.*

Proof. Let W^i_j be the largest value in some row i. Then, if we take a corresponding matching of size W^i_j, it is easy to see that we can remove the last two edges and replace them by an arbitrary edge (i, k). This proves that $W^i_k \ge W^i_j - 1$.

More formally, we can argue by the recursion (5): Assume there are values $W^i_k \le W^i_j - 2$ in row i. By Lemma 4 we can assume $k < j$. By (5), $W^i_j = 1 + W^{i'}_{j'} = 2 + W^{i''}_{j''}$ with $R^{i''} < j' < L^i$

Algorithm 2: Unweighted Maximum Matching, initial version

```
Set Q_1 := Q_2 := ··· := Q_{n_U} := 0
F := 0
for ℓ := 1 to n_V do
    for all rows i ∈ U with L^i = ℓ do ▷ Process each interval i that starts at ℓ
        w := F + 1 ▷ leftmost entry
        t_w := leftmost endpoint R^{i'} of a row i' that contains an entry W^{i'}_j = w with
         j < L^i ≡ ℓ
        if t_w < R^i then ▷ There are two values w and w + 1 in this row:
            ▷ W^i_j = w        for j = L^i, ..., t_w
            ▷ W^i_j = w + 1    for j = t_w + 1, ..., R^i
            Q_{R^i} := max{Q_{R^i}, w + 1} ▷ The largest entry is w + 1.
        else ▷ The same entry w is used for the whole row.
            Q_{R^i} := max{Q_{R^i}, w} ▷ The largest entry is w.
    F := max{F, Q_ℓ} ▷ update F as ℓ advances
return F
```

for some $i'' < i' < i$. Thus, $j'' \leq R^{i''} < j' < L^i \leq k$ and by definition of W^i_k according to (5) we have $W^i_j - 2 = W^{i''}_{j''} < W^i_k \leq W^i_j - 2$, which is a contradiction. □

Specializing Algorithm 1 to the unweighted case leads to a solution with $O(m)$ running time. Our $O(n)$-time algorithm will follow the general scheme of Algorithm 1, with the following modifications.

- In view of Lemmas 4 and 5, we will not fill each row individually, but we will just determine the leftmost value w and the position where the entries switch from w to $w + 1$ (if any).
- The computation of the leftmost entry is exactly as in Algorithm 1.
- The position where the entries of row i switch from w to $w+1$ can be determined from (5): If there is a row i' containing an entry w left of L^i, then W^i_j must be $w + 1$ as soon as $j > R^{i'}$. The algorithm determines the *threshold position* t_w as the smallest right endpoint $R^{i'}$ under these constraints. Then the entries $w + 1$ in row i start at $j = t_w + 1$ if these entries are still part of the row.
- We do not maintain the whole array P_r for each r, but only its last entry $P_r[r]$; this is sufficient for updating F and thus for computing the leftmost entries in the rows. We call this value Q_r.

This leads to Algorithm 2.

We will improve Algorithm 2 by maintaining the values t_w instead of computing them from scratch. We use the fact that the smallest value w in the row is known, and hence we can associate t_w with the value w instead of the row index i, as is already apparent from our chosen notation. We update t_w whenever ℓ increases. The details are shown in Algorithm 3. The differences to Algorithm 2 are marked by $\triangle$.

This still does not achieve $O(n)$ running time. The final improvement comes from realizing that it is sufficient to update t_w when $W^{i'}_l$ is the *leftmost* entry w in row i'. The time when such an update occurs can be predicted when a row is generated. To this end, we maintain a list $\mathcal{T}_j$ for $j = 1, \ldots, n_V$ that records the updates that are due when ℓ becomes j. This final version is Algorithm 4.

The runtime of Algorithm 4 is $O(n_U + n_V)$: Processing each interval i takes constant time and adds at most two pairs to the lists $\mathcal{T}$. Thus, processing the lists $\mathcal{T}$ for updating the t_w array takes also only $O(n_U)$ time.

Algorithm 3: Unweighted Maximum Matching, second version

```
△ Set t_1 := t_2 := ··· := t_{n_U} := n_V + 1 ▷ The value n_V + 1 acts like ∞.
  Set Q_1 := Q_2 := ··· := Q_{n_U} := 0
  F := 0
  for ℓ := 1 to n_V do
      for all rows i ∈ U with L^i = ℓ do ▷ Process each interval i that starts at ℓ
          w := F + 1 ▷ leftmost entry
△         ▷ t_w is no longer computed from scratch
          if t_w < R^i then ▷ There are two values w and w + 1 in this row:
              ▷ W^i_j = w        for j = L^i, ..., t_w,
              ▷ W^i_j = w + 1    for j = t_w + 1, ..., R^i.
              Q_{R^i} := max{Q_{R^i}, w + 1} ▷ The largest entry is w + 1.
          else ▷ The same entry w is used for the whole row.
              Q_{R^i} := max{Q_{R^i}, w} ▷ The largest entry is w.
      F := max{F, Q_ℓ} ▷ update F as ℓ is incremented
△     for all entries W^{i'}_ℓ in column ℓ do
△         w := W^{i'}_ℓ
△         t_w := min{t_w, R^{i'}};
  return F
```

Some simplifications are possible: The addition of (w, R^i) to the list $\mathcal{T}_\ell$ in the case of two values can actually be omitted, as it leads to no decrease in t_w: t_w is already $< R^i$. The algorithm could be further streamlined by observing that at most two consecutive values of t_w need to be remembered at any time.

Again, it is easy to modify the algorithm to return a maximum induced matching in addition to its size.

Theorem 2. *Given a compact representation, a maximum-cardinality induced matching of a convex bipartite graph can be computed in $O(n)$ time.*

4 Minimum Chain Covers

In convex bipartite graphs, the size of a maximum-cardinality induced matching equals the number of chain subgraphs of a minimum chain cover [24]. In this section we use this duality and extend our Algorithm 4 to obtain a minimum chain cover of a convex bipartite graph $G = (U, V, E)$.

Let W^* be the cardinality of a maximum induced matching of G. Accordingly, the values W^i_j cover the range $\{1, \ldots, W^*\}$. We create W^* chain subgraphs $Z_1, \ldots, Z_{W^*}$ of G. The edges (i, j) with $W^i_j = w$ will be part of the chain subgraph Z_w.

As already observed in [24], the edges with a fixed value of W^i_j may contain independent edges and, thus, do not necessarily constitute a chain graph. Accordingly, Yu, Chen, and Ma [24] describe a strategy to extend the edge set for each value of $W^i_j = w$ to a chain graph Z_w. Their original implementation runs in time $O(m^2)$. Brandstädt, Eschen, and Sritharan [3] give an improved implementation with runtime $O(n^2)$. We implement their strategy in $O(n)$ time, given a compact representation. The correctness was already shown in [24]. We give a new independent proof. The following characterization is often used as an alternative definition of chain graphs:

Lemma 6. *A bipartite graph $(\bar{U}, \bar{V}, \bar{E})$ is a chain graph if and only if the sets of neighbors $\bar{V}(i) := \{\, j \in \bar{V} \mid (i, j) \in \bar{E} \,\}$ of the vertices $i \in \bar{U}$ form a chain in the inclusion order. (Equal sets are allowed.) In other words, among any two sets $\bar{V}(i)$ and $\bar{V}(i')$, one must be contained in the other.*

Algorithm 4: Unweighted Maximum Matching, final version

△ Initialize lists $\mathcal{T}_1, \dots, \mathcal{T}_{n_V}$ to empty lists
Set $t_1 := t_2 := \cdots := t_{n_U} := n_V + 1$
Set $Q_1 := Q_2 := \cdots := Q_{n_U} := 0$
$F := 0$
for $\ell := 1$ **to** n_V **do**
  for all *rows* $i \in U$ *with* $L^i = \ell$ **do** ▷ Process each interval i that starts at ℓ
    $w := F + 1$ ▷ leftmost entry
    if $t_w < R^i$ **then** ▷ There are two values w and $w+1$ in this row:
      ▷ $W^i_j = w$ for $j = L^i, \dots, t_w$,
      ▷ $W^i_j = w + 1$ for $j = t_w + 1, \dots, R^i$.
△   add $(w+1, R^i)$ to the list $\mathcal{T}_{t_w+1}$ ▷ don't forget to update t_{w+1} when ℓ reaches $t_w + 1$
△   add (w, R^i) to the list $\mathcal{T}_\ell$ ▷ don't forget to update t_w when ℓ advances
      $Q_{R^i} := \max\{Q_{R^i}, w + 1\}$
    else ▷ The same entry w is used for the whole row.
△   add (w, R^i) to the list $\mathcal{T}_\ell$
      $Q_{R^i} := \max\{Q_{R^i}, w\}$
  $F := \max\{F, Q_\ell\}$ ▷ update F as ℓ advances
△ **for all** $(w, r) \in \mathcal{T}_\ell$ **do** $t_w := \min\{t_w, r\}$ ▷ perform the necessary updates
return F

Proof. This is a direct consequence of the fact that edges (i, j) and (i', j') are independent if and only if $j' \notin \bar{V}(i)$ and $j \notin \bar{V}(i')$. ☐

The condition that the neighborhoods must form a chain is apparently the reason for calling these graphs *chain graphs*, however, we did not find a reference for this.

We use U_w to denote the set of rows that contain entries $W^i_j = w$. For every row $i \in U_w$, we determine the beginning and ending points B^i_w, E^i_w with this color, that is, $W^i_j = w \iff B^i_w \le j \le E^i_w$. We extend every such interval $[B^i_w, E^i_w]$ to the left by choosing a new starting point $\hat{B}^i_w$ according to the formula

$$\hat{B}^i_w := \min\{B^i_w\} \cup \{\, B^{i'}_w \mid i' \in U_w,\ E^{i'}_w < E^i_w \,\} \tag{6}$$
$$= \min\{B^i_w\} \cup \{\, \hat{B}^{i'}_w \mid i' \in U_w,\ E^{i'}_w < E^i_w \,\} \tag{7}$$

The second expression uses the new values $\hat{B}$ on the right-hand side. It is easy to see that the two expressions are equivalent: Using (6) for the definition of $\hat{B}^{i'}_w$, the expression (7) becomes

$$\min\{B^i_w\} \cup \{\, B^{i'}_w \mid i' \in U_w,\ E^{i'}_w < E^i_w \,\} \cup \{B^{i''}_w \mid i'' \in U_w, E^{i''}_w < E^{i'}_w < E^i_w, i' \in U_w\}. \tag{8}$$

The third set is contained in the second set, and thus, (8) is equal to $\hat{B}^i_w$ according to (6).

We construct the chain graph Z_w as the graph with the extended intervals $[\hat{B}^i_w, E^i_w]$. Figure 4 shows an example. It is obvious by construction that these intervals satisfy the conditions of a chain graph: By Lemma 6, we have to show that there are no two intervals $[\hat{B}^i_w, E^i_w]$, $[\hat{B}^{i'}_w, E^{i'}_w]$ with $\hat{B}^{i'}_w < \hat{B}^i_w$ and $E^{i'}_w < E^i_w$. But if the last condition holds, (7) ensures that $\hat{B}^i_w \le \hat{B}^{i'}_w$.

The only thing that could go wrong is that $\hat{B}^i_w$ becomes too small so that the chain graph is not a subgraph of G. The following lemma shows that this is not the case.

Lemma 7. *$\hat{B}^i_w \ge L^i$ for every $i \in U_w$.*

Proof. For the sake of contradiction, assume $\hat{B}^i_w < L^i$. By (6), there is a row $i' \in U_w$ such that $B^{i'}_w < L^i$ and $E^{i'}_w < E^i_w$. Setting $j = E^i_w$ and $j' = B^{i'}_w$ in the recursion (5), we conclude that

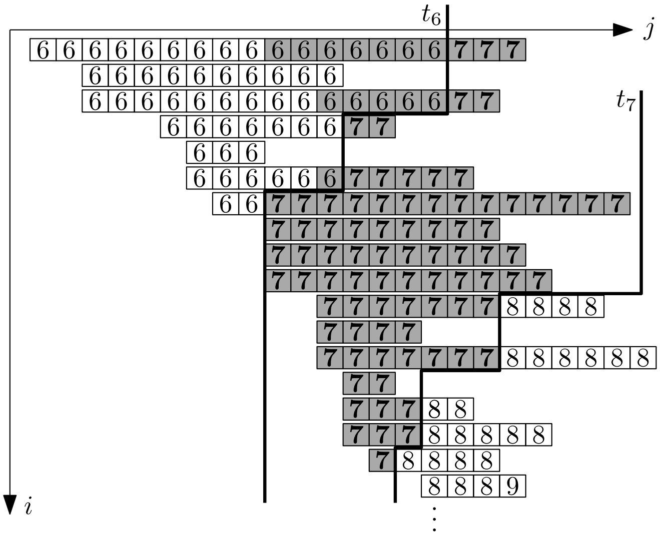


Fig. 4: An example showing a section of the computation of W_j^i by Algorithm 4. The threshold values t_6 and t_7 are shown as they change with the rows that are successively considered. The shaded entries form the chain subgraph Z_7 that is used for the chain cover.

$E_w^i \le R^{i'}$, because otherwise, (5) would imply $w = W^i_{E_w^i} \ge 1 + W^{i'}_{B_w^{i'}} = 1 + w$. Thus, (i', E_w^i) is an edge of G. By Lemma 5, $W^{i'}_{E_w^i} = w + 1$. By (5), there is an edge (i'', j'') with $W^{i''}_{j''} = w$, $R^{i''} < E_w^i$ and $j'' < L^{i'} < L^i$. Again by (5), such an edge (i'', j'') would imply that $W^i_{E_w^i} \ge w+1$, a contradiction. $\square$

Algorithm 5 carries out the computation of (6). It processes the triplets (B_w^i, E_w^i, w) in increasing order of the endpoints $E_w^i = r$. This can be done in linear time, by first sorting the $O(n_U)$ triples (B_w^i, E_w^i, w) into n_V buckets according to the value of E_w^i. Thus, Algorithm 5 takes linear time $O(n)$. By Lemma 6, the result is a chain cover, which by duality is minimum. Each row belongs to at most two chain subgraphs, and thus the chain cover consists of at most $2n_U$ such row intervals in total. It is straightforward to extend Algorithm 4 to compute the sets U_w and the quantities B_w^i, E_w^i, and thus the cover can be constructed in $O(n)$ time in compressed form.

Theorem 3. *Given a compact representation of a convex bipartite graph, a compact representation of a minimum chain cover can be computed in $O(n)$ time.*

Given a compact representation of a minimum chain cover, we can list all the edges of its chain subgraphs in $O(n+m)$ time since every edge is contained in at most two chain subgraphs. As mentioned in the introduction, a compact representation of a convex bipartite graph can be computed in $O(n+m)$ time [20, 22, 2]. Thus, Algorithm 4 and Algorithm 5 can also be used to obtain:

Theorem 4. *A minimum chain cover of a convex bipartite graph can be computed in $O(n+m)$ time.*

5 Certification of Optimality

An induced matching H together with a chain cover of the same cardinality provides a *certificate of optimality*, of size $O(n)$. As we will establish in the following discussion, it is easy to *check* this certificate for validity in linear time. This is easier than *constructing* the largest induced matching with our algorithm. Thus, it is possible to establish correctness of the result beyond

Algorithm 5: Constructing a chain graph $\{\, (i,j) \mid i \in U_w,\, \hat{B}^i_w \le j \le E^i_w \,\}$, $1 \le w \le W^*$

```
▷ U_w := { i ∈ U | row i contains an entry w }
▷ Let B^i_w and E^i_w such that in row i, the entries with W^i_j = w are those with
  B^i_w ≤ j ≤ E^i_w
Set G_1 := G_2 := ··· := G_{W*} := n_V + 1 ▷ The value n_V + 1 acts like ∞
for r := 1 to n_v do
    ▷ We maintain the quantities G_w ≡ min{B^i_w | E^i_w < r} for w = 1, ..., W*.
    for all (B^i_w, E^i_w, w) with E^i_w = r do
        B̂^i_w := min{B^i_w, G_w}
    for all (B^i_w, E^i_w, w) with E^i_w = r do ▷ update G_w for the increment of r
        G_w := min{B^i_w, G_w}
```

doubt, for each particular instance of the problem, without having to trust the correctness of our algorithms and their implementations, see [17] for a survey about this concept.

It is trivial to check whether the matching H is contained in the graph. To test whether it forms an induced matching, we sort the edges (i,j) by j. This takes $O(n)$ time with bucket-sort. Then, by Lemma 1, it is sufficient to test consecutive edges for independence, and each such test takes only constant time according to Observation 1.

To establish the validity of a chain cover $\{Z_1, \dots, Z_{W^*}\}$, we need to check that the edges of G are covered and each Z_w is a chain subgraph. The chain subgraphs $Z_w = \{\, (i,j) \mid i \in U_w,\, \hat{B}^i_w \le j \le E^i_w \,\}$, for $1 \le w \le W^*$ are compactly represented by a set of at most $2n_U$ quadruples $(w, i, \hat{B}^i_w, E^i_w)$. The following checking procedure works in linear time for any chain cover as long as it consists of convex bipartite subgraphs. It does not use any special properties of the cover produced by our algorithm.

We sort the quadruples $(w, \hat{B}^i_w, -E^i_w, i)$ lexicographically. Then it is easy to check the chain graph property using the characterization of Lemma 6: The intervals $[\hat{B}^i_w, E^i_w]$ that belong to a fixed chain graph Z_w (these are consecutive in the list) ought to be nested. Since the starting points $\hat{B}^i_w$ are weakly increasing, this amounts to checking that the endpoints E^i_w decrease weakly.

To check that the chain graphs are contained in G and they collectively cover G, we sort the quadruples $(i, \hat{B}^i_w, E^i_w, w)$. The union of the intervals $[\hat{B}^i_w, E^i_w]$ that are the neighbors of a fixed vertex $i \in U$ (these are consecutive in the list) can be incrementally formed, and the resulting interval is compared against $[L^i, R^i]$. As soon as a gap would form in this union, we can abort the test, since the intervals are sorted by left endpoint and it is then impossible to form a connected interval $[L^i, R^i]$.

The required lexicographic sorting operations can be carried out in $O(n)$ time by bucket-sort.

6 Outlook: Duality

The existence of a pair of maximum induced matchings and smallest chain covers with the same size is a manifestation of strong duality between independents sets and clique covers in perfect graphs. We mentioned in the introduction that our maximum induced matching problem is an instance of a maximum independent set problem in the square of a line graph, and the chain cover is a covering by cliques. Yu, Chen and Ma [24] established that the square of the line graph of a convex bipartite graph is a co-comparability graph. Therefore, it is also a perfect graph. It follows that the linear program for maximizing the size of an induced matching is totally dual integral. As a corollary of this fact, we recover our strong duality result: the existence of a *primal* optimal solution (maximum induced matching) and a *dual* optimal solution (smallest chain cover) with matching objective function values.

This duality relation for perfect graphs extends to the weighted version. Thus, there should also be a weighted chain cover with the same weight as the maximum weight of an induced matching. It would be interesting to extend our primal Algorithm 1 in weighted graphs to a fast combinatorial algorithm for finding minimum-weight chain covers, as Algorithm 5 does for the unweighted version.

References

[1] K. Asdre, S. D. Nikolopoulos: NP-completeness results for some problems on subclasses of bipartite and chordal graphs. Theoretical Computer Science 381(1), 248–259 (2007), DOI:10.1016/j.tcs.2007.05.012

[2] K. S. Booth, G. S. Lueker: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comput. Syst. Sci. 13(3), 335–379 (1976), DOI:10.1016/S0022-0000(76)80045-1

[3] A. Brandstädt, E. M. Eschen, R. Sritharan: The induced matching and chain subgraph cover problems for convex bipartite graphs. Theor. Comput. Sci. 381(1–3), 260–265 (2007), DOI:10.1016/j.tcs.2007.04.006

[4] K. Cameron: Induced matchings. Discrete Applied Mathematics 24(1–3), 97–102 (1989), DOI:10.1016/0166-218X(92)90275-F

[5] K. Cameron, R. Sritharan, Y. Tang: Finding a maximum induced matching in weakly chordal graphs. Discrete Mathematics 266(1-3), 133–142 (2003), DOI:10.1016/S0012-365X(02)00803-8

[6] J.-M. Chang: Induced matchings in asteroidal triple-free graphs. Discrete Applied Mathematics 132(1–3), 67–78 (2003), DOI:10.1016/S0166-218X(03)00390-1

[7] G. Ding: Covering the edges with consecutive sets. Journal of Graph Theory 15(5), 559–562 (1991), DOI:10.1002/jgt.3190150508

[8] W. Duckworth, D. Manlove, M. Zito: On the approximability of the maximum induced matching problem. J. Discrete Algorithms 3(1), 79–91 (2005), DOI:10.1016/j.jda.2004.05.001

[9] G. Gallo: An $O(n \log n)$ algorithm for the convex bipartite matching problem. Operations Research Letters 3(1), 31–34 (1984), DOI:10.1016/0167-6377(84)90068-3

[10] F. Glover: Maximum matching in a convex bipartite graph. Naval Research Logistics Quarterly 14(3), 313–316 (1967), DOI:10.1002/nav.3800140304

[11] M. C. Golumbic, M. Lewenstein: New results on induced matchings. Discrete Applied Mathematics 101(1-3), 157–165 (2000), DOI:10.1016/S0166-218X(99)00194-8

[12] P. L. Hammer, U. N. Peled, X. Sun: Difference graphs. Discrete Applied Mathematics 28(1), 35–44 (1990), DOI:10.1016/0166-218X(90)90092-Q

[13] R. Hung: Linear-time algorithm for the paired-domination problem in convex bipartite graphs. Theory Comput. Syst. 50(4), 721–738 (2012), DOI:10.1007/s00224-011-9378-8

[14] I. Katriel: Matchings in node-weighted convex bipartite graphs. INFORMS Journal on Computing 20(2), 205–211 (2008), DOI:10.1287/ijoc.1070.0232

[15] D. Kobler, U. Rotics: Finding maximum induced matchings in subclasses of claw-free and P_5-free graphs, and in graphs with matching and induced matching of equal maximum size. Algorithmica 37(4), 327–346 (2003), DOI:10.1007/s00453-003-1035-4

[16] V. V. Lozin: On maximum induced matchings in bipartite graphs. Inf. Process. Lett. 81(1), 7–11 (2002), DOI:10.1016/S0020-0190(01)00185-5

[17] R. M. McConnell, K. Mehlhorn, S. Näher, P. Schweitzer: Certifying algorithms. Computer Science Review 5(2), 119–161 (May 2011), DOI:10.1016/j.cosrev.2010.09.009

[18] H. Müller: Hamiltonian circuits in chordal bipartite graphs. Discrete Mathematics 156(1–3), 291–298 (1996), DOI:10.1016/0012-365X(95)00057-4

[19] A. Pandey, B. S. Panda, P. Dane, M. Kashyap: Induced matching in some subclasses of bipartite graphs. In: D. R. Gaur, N. S. Narayanaswamy (eds.) Algorithms and Discrete Applied Mathematics – Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16–18, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10156, pp. 308–319. Springer (2017), DOI:10.1007/978-3-319-53007-9_27

[20] J. Soares, M. A. Stefanes: Algorithms for maximum independent set in convex bipartite graphs. Algorithmica 53(1), 35–49 (2009), DOI:10.1007/s00453-007-9006-9

[21] J. P. Spinrad: Efficient Graph Representations. American Mathematical Society (2003)

[22] G. Steiner, J. Yeomans: A linear time algorithm for maximum matchings in convex, bipartite graphs. Computers & Mathematics with Applications 31(12), 91–96 (1996), DOI:10.1016/0898-1221(96)00079-X

[23] L. J. Stockmeyer, V. V. Vazirani: NP-completeness of some generalizations of the maximum matching problem. Inf. Process. Lett. 15(1), 14–19 (1982), DOI:10.1016/0020-0190(82)90077-1

[24] C.-W. Yu, G.-H. Chen, T.-H. Ma: On the complexity of the k-chain subgraph cover problem. Theor. Comput. Sci. 205(1–2), 85–98 (1998), DOI:10.1016/S0304-3975(97)00036-4