A Linear Time Algorithm to Compute a Maximum Weighted Independent Set on Cocomparability Graphs

Ekkehard Köhler[a], Lalla Mouatadid[b]

[a]*Brandenburg University of Technology, 03044 Cottbus, Germany*
[b]*University of Toronto, Toronto ON M5S 2J7, Canada*

Abstract

The maximum weight independent set (WMIS) problem is a well-known NP-hard problem. It is a generalization of the maximum cardinality independent set problem where all the vertices have identical weights. There is a $\mathcal{O}(n^2)$ time algorithm to compute a WMIS for cocomparability graphs by computing a maximum weight clique on the corresponding complement of the graph [1]. We present the first $\mathcal{O}(m+n)$ time algorithm to compute a WMIS directly on the given cocomparability graph, where m and n are the number of edges and vertices of the graph respectively. As a corollary, we get the minimum weight vertex cover of a cocomparability graph in linear time as well.

Keywords: maximum weight independent set, cocomparability graphs, posets, minimum weight vertex cover

---

1. Introduction

Given a graph $G(V,E)$, an *independent set* (also called *stable set*) $I \subseteq V$, is a subset of pairwise non-adjacent vertices. For $G(V,E,w)$ being a graph together with a weight function $w : V \to \mathbb{R}$, the weighted maximum independent set (WMIS) problem asks for an independent set $I \subseteq V$ such that $\sum_{v \in I} w(v)$ is maximum. This problem is a generalization of the maximum cardinality independent set problem where all vertices have equal weights. The WMIS problem has been widely studied as it naturally arises in different applications, such as scheduling [2], combinatorial auctions [3], molecular

Email addresses: `ekkehard.koehler@b-tu.de` (Ekkehard Köhler), `lalla@cs.toronto.edu` (Lalla Mouatadid)

Preprint submitted to Elsevier

biology [4] to name a few. The problem is NP-hard for arbitrary graphs; we restrict ourselves to the class of cocomparability graphs and present a linear time algorithm for this case.

Let $G(V, E)$ be a graph where $n = |V|$ and $m = |E|$, and let $N(v)$ (resp. $N[v]$) denote the open (resp. closed) neighbourhood of vertex v; $N(v) = \{u \in V \mid uv \in E\}$ and $N[v] = N(v) \cup \{v\}$. A graph $G(V, E)$ is a *cocomparability* graph if its complement is a *comparability* graph. A graph $G(V, E)$ is a comparability graph if E admits an acyclic transitive orientation. That is, if $uv, vw \in E$, and they are oriented $u \to v$, and $v \to w$ then uw has to be contained in E and must be oriented $u \to w$. Cocomparability graphs are a subfamily of perfect graphs and have been well studied. Many problems on this graph class are solved by computing the complement of the given graph, and translating the problem into its complement problem on comparability graphs. This transformation necessitates $\Omega(n^2)$ computation, whereas for some problems direct solutions in $\mathcal{O}(n + m)$ are possible. Finding a WMIS in a cocomparability graph, for example, is equivalent to finding a maximum weighted clique in its complement. There exists a linear time dynamic programming algorithm to compute the maximum weight clique on a comparability graph, given a transitive orientation of the edges [1]. This implies an $\mathcal{O}(n^2)$ time algorithm to compute a WMIS on a cocomparability graph.

The idea to solve problems directly on cocomparability graphs instead of going over to the complement graph has been around for a while and a number of problems have been solved in this way, such as domination [5] and the minimum feedback vertex set problem [6]. Recently, there have been new approaches for solving problems directly on the given cocomparability graph. In [7] for instance, Mertzios and Corneil presented the first polynomial time algorithm to solve the longest path problem on cocomparability graphs, and in [8] Corneil et al. gave the first near linear time certifying algorithm to compute a minimum path cover, and thus a Hamilton path (if one exists), directly on cocomparability graphs. Motivated by this idea, we present the first linear time algorithm to compute a WMIS directly on a cocomparability graph. The unweighted case has been known to take $\mathcal{O}(m + n)$ time [9]. As a corollary to our result, we also get the minimum weight vertex cover of a cocomparability graph in linear time.

Cocomparability graphs have a vertex ordering characterization, known as a *cocomparability order* σ, or an *umbrella-free* order; more precisely, an ordering $\sigma = v_1 \prec_\sigma v_2 \prec_\sigma \cdots \prec_\sigma v_n$ is a cocomparability order iff for any triple $u \prec_\sigma v \prec_\sigma w$ with $uw \in E$, either $uv \in E$ or $vw \in E$ or both [5]. In other words, σ does not contain an *umbrella*, which is a triple of vertices $u \prec_\sigma v \prec_\sigma w$ with $uw \in E$ but $uv, vw \notin E$. In [10], McConnell and Spinrad presented an algorithm to compute such an ordering in $\mathcal{O}(m+n)$ time. We use their algorithm, denoted as $\sigma \leftarrow ccorder(G)$ to compute such an ordering.

This paper is organized as follows. In Section 2 we present an overview of the algorithm, followed by its formal description and in Section 3, we prove the correctness of the algorithm, present implementation details and the complexity analysis.

2. The Algorithm

Let $G(V, E, w)$ be a weighted cocomparability graph and let $X \subseteq V$ be the subset of vertices with non-positive weight, i.e., $X = \{v : w(v) \leq 0\}$. Any vertex $v \in X$ that belongs to an independent set S will not increase the total weight of S. Therefore if $X \neq \emptyset$, we can restrict ourselves to $G[V \backslash X]$, which is also a cocomparability graph that can easily be computed in $\mathcal{O}(m+n)$ time.

Suppose $G(V, E, w)$ is a cocomparability graph with positive weight function $w : V \to \mathbb{R}_{>0}$. Using the algorithm in [10], we compute a cocomparability order σ of V in $\mathcal{O}(m+n)$ time where $\sigma = v_1 \prec_\sigma v_2 \prec_\sigma \cdots \prec_\sigma v_n$. We then construct a new permutation τ of the vertices as follows: we process one vertex at a time according to the order imposed by σ from left to right. To each v_i we associate an updated weight $\tilde{w}(v_i)$ and an [independent] set $\mathcal{S}_{v_i}$ (containing v_i) of total weight $\tilde{w}(v_i)$. The vertices from v_1 to v_i are then reordered such that the new ordering is non-decreasing with respect to their updated weights $\tilde{w}$; τ_i denotes the resulting permutation on the processed vertices $v_1, \ldots, v_i$. In other words, for vertices v_k, v_j ($1 \leq k, j \leq i$, $k \neq j$),

$$\text{if} \;\; v_k \prec_{\tau_i} v_j \;\; \text{then} \;\; \tilde{w}(v_k) \leq \tilde{w}(v_j). \tag{1}$$

Initially τ_1 is just $\{v_1\}$, $\tilde{w}(v_1) = w(v_1)$, and $\mathcal{S}_{v_1} = \{v_1\}$. For every vertex v_i ($i > 1$), we scan through τ_{i-1} from right to left, looking for the rightmost non-neighbour of v_i. Let u denote such a vertex (if it exists); $\tilde{w}(v_i)$ and $\mathcal{S}_{v_i}$ are then set to

$$\begin{aligned} \tilde{w}(v_i) &= w(v_i) + \tilde{w}(u) \\ \mathcal{S}_{v_i} &= \{v_i\} \cup \mathcal{S}_u. \end{aligned}$$

If no such vertex u exists, then

$$\begin{aligned}\tilde{w}(v_i) &= w(v_i)\\ \mathcal{S}_{v_i} &= \{v_i\}.\end{aligned}$$

τ_i is the permutation of $\{v_1, \ldots, v_i\}$ created by inserting v_i into τ_{i-1} such that (1) holds and thus preserving the non-decreasing order of the updated weights. Since the weights are strictly positive, it is easy to see that $\tilde{w}(v_i) = w(v_i) + \tilde{w}(u)$ implies $\tilde{w}(v_i) > \tilde{w}(u)$ and thus also implies $u \prec_{\tau_i} v_i$.

Notice that if there exists a vertex x in τ_{i-1} such that $\tilde{w}(x) = \tilde{w}(v_i)$, then v_i is inserted to the right of vertex x in τ_{i-1}. We say that a vertex v_i has been *processed* as soon as it is inserted into τ_{i-1} and thus τ_i is created. When all vertices are processed, we have determined τ_n. We return $\mathcal{S}_z$ as a maximum weight independent set of G and $\tilde{w}(z)$ as its corresponding total weight, where z is the rightmost vertex in τ_n.

We now present the formal description of the algorithm; recall that $ccorder(G)$ is the procedure presented in [10] to compute a cocomparability order in $\mathcal{O}(m + n)$ time.

Algorithm 1: CCWMIS

Input: $G = (V, E, w)$, $w : V \to \mathbb{R}_{>0}$
Output: A maximum weight independent set together with its weight

$\sigma \leftarrow ccorder(G(V, E))$; // $\sigma = (v_1, v_2, \ldots, v_n)$
for $i \leftarrow 1$ **to** n **do**
  $\tilde{w}(v_i) \leftarrow w(v_i)$;
  $\mathcal{S}_{v_i} \leftarrow \{v_i\}$;
$\tau_1 \leftarrow (v_1)$; // Constructing τ_i
for $i \leftarrow 2$ **to** n **do**
  Choose u to be rightmost non-neighbour of v_i with respect to τ_{i-1};
  if u *exists* **then**
    $\tilde{w}(v_i) \leftarrow w(v_i) + \tilde{w}(u)$;
    $\mathcal{S}_{v_i} \leftarrow \{v_i\} \cup \mathcal{S}_u$;
  $\tau_i \leftarrow insert(v_i, \tau_{i-1})$;
  // Insert v_i into τ_{i-1} such that τ_i stays ordered with respect to $\tilde{w}(\cdot)$
$z \leftarrow$ the rightmost vertex in τ_n;
return $\mathcal{S}_z$ *and* $\tilde{w}(z)$;

We illustrate the algorithm using a cocomparability graph and a corresponding cocomparability ordering given in Figure 1. Table 1 shows how τ_i is created by the algorithm. Recall that the vertices are processed in σ's order and vertex v_i is inserted into τ_{i-1} according to its updated weight.

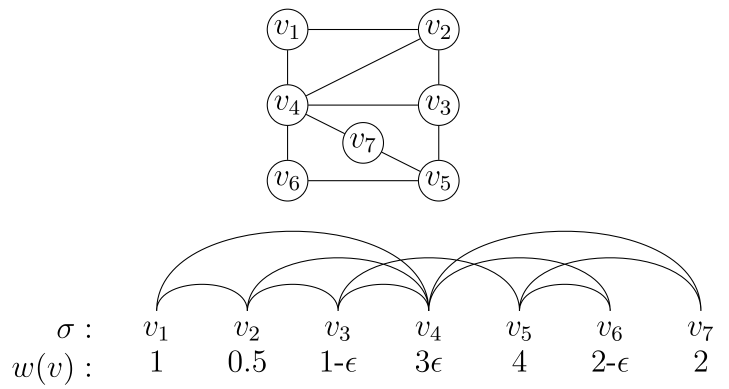

Figure 1: A cocomparability graph with a valid cocomparability ordering; positive weights are given below the vertices, with $0 < \epsilon < \frac{1}{6}$.

v_i	u	$\mathcal{S}_{v_i}$	$\tilde{w}(v_i)$	τ_i
v_1	-	$\{v_1\}$	1	v_1
v_2	-	$\{v_2\}$	0.5	v_2, v_1
v_3	v_1	$\{v_3, v_1\}$	$2 - \epsilon$	v_2, v_1, v_3
v_4	-	$\{v_4\}$	3ϵ	v_4, v_2, v_1, v_3
v_5	v_1	$\{v_5, v_1\}$	5	v_4, v_2, v_1, v_3, v_5
v_6	v_3	$\{v_6, v_3, v_1\}$	$4 - 2\epsilon$	$v_4, v_2, v_1, v_3, v_6, v_5$
v_7	v_6	$\{v_7, v_6, v_3, v_1\}$	$6 - 2\epsilon$	$v_4, v_2, v_1, v_3, v_6, v_5, v_7$

Table 1: Step by step construction of the ordering τ_n as computed by Algorithm 1. At iteration i, v_i is being processed to create τ_i; u denotes the rightmost non-neighbour of the vertex being processed; "-" means no such vertex u exists. By Algorithm 1, $z = v_7$ and thus a maximum weight independent set of the graph in Figure 1 is $\mathcal{S}_z = \{v_7, v_6, v_3, v_1\}$ with weight $\tilde{w}(z) = 6 - 2\epsilon$.

3. Correctness, Complexity Analysis, and Robustness

Recall that $\mathcal{S}_{v_i}$ is the set associated with v_i recursively constructed by finding u, the rightmost non-neighbour of v_i in τ_{i-1}; in other words $\mathcal{S}_{v_i}$ denotes a set of vertices including v_i whose weights sum up to $\tilde{w}(v_i)$. Therefore $w(\mathcal{S}_{v_i}) = \tilde{w}(v_i)$. For all i, $\mathcal{S}_{v_i}$ is initialized to $\{v_i\}$ in step 4 of Algorithm 1 and is updated accordingly in step 10.

Lemma 1. *For all i, on entry to step 11 of Algorithm 1, the set $\mathcal{S}_{v_i}$ is an independent set.*

Proof. The proof is by induction on i. For $i = 1$ the set $\mathcal{S}_{v_1} = \{v_1\}$ as initialized in step 4 is an independent set.

Suppose the lemma holds for all $j \in \{1, \dots, i-1\}$ and look at vertex v_i. Obviously, if there is no u as defined in step 7, then v_i is universal to the vertices in τ_{i-1} and thus we have $\mathcal{S}_{v_i} = \{v_i\}$ as initialized in step 4. Consider now the case that there is such a vertex u and assume for contradiction that i is the first iteration where the set $\mathcal{S}_{v_i}$ computed in step 10 is not an independent set. At iteration i, v_i is being processed; let $v_{j<i}(= u)$ be the rightmost non-neighbour of v_i in τ_{i-1} which means v_j was processed before v_i and thus:

$$
\begin{aligned}
& v_j \prec_\sigma v_i \\
& v_j v_i \notin E \\
& \mathcal{S}_{v_i} = \{v_i\} \cup \mathcal{S}_{v_j} && (2) \\
& \mathcal{S}_{v_j} \text{ is an independent set by the induction hypothesis.} && (3)
\end{aligned}
$$

Given (2) and (3), if $\mathcal{S}_{v_i}$ is not an independent set, there must exist a vertex $a \in \mathcal{S}_{v_j}$ where $av_i \in E$, and by (3), $av_j \notin E$. Furthermore, we know that $a \prec_\sigma v_j$, since for creating τ_j vertex v_j was inserted into τ_{j-1} to the right of its rightmost non-neighbour in τ_{j-1}. Thus the ordering of the triple (a, v_j, v_i) implied by the cocomparability ordering σ is $a \prec_\sigma v_j \prec_\sigma v_i$. However, the edge av_i flying over v_j contradicts σ being a cocomparability order; therefore on entry to step 11 of Algorithm 1, $\mathcal{S}_{v_i}$ is an independent set. □

Lemma 2. *For all i, on entry to step 11 of Algorithm 1, in the graph $G[v_1, \dots, v_i]$ the set $\mathcal{S}_{v_i}$ is of maximum weight among the independent sets containing v_i.*

Proof. The proof is again by induction on i. For $i = 1$, the maximum weight independent set in $G[v_1]$ is just $\mathcal{S}_{v_1} = \{v_1\}$ with $\tilde{w}(v_1) = w(v_1)$.

Suppose now that the claim holds for all $j \in \{1, \dots, i-1\}$ and let v_i be the vertex considered. Further, let u be the rightmost vertex that is non-adjacent to v_i in τ_{i-1}. If no such vertex u exists, then v_i is universal to all vertices in τ_{i-1}, and $\mathcal{S}_{v_i} = \{v_i\}$ as initialized in step 4 is the only independent set in $G[v_1, \ldots, v_i]$ that contains v_i, and thus has maximum weight among all independent sets containing v_i. Suppose now that such a vertex u exists. Since τ_{i-1} contains a non-decreasing order of the updated weights $\tilde{w}(v)$ for $v \in \{v_1, \ldots, v_{i-1}\}$, we thus know that $\mathcal{S}_u$ is an independent set containing u with the maximum weight such that v_i can be added to $\mathcal{S}_u$ and by Lemma 1 maintains independency. If there were another independent set $\mathcal{S}_a$ for $a \in \{v_1, \ldots, v_{i-1}\}$ such that $\tilde{w}(a) > \tilde{w}(u)$ and a is non-adjacent to v_i, then $u \prec_{\tau_{i-1}} a$ thereby contradicting Algorithm 1 choosing u. Therefore $\mathcal{S}_{v_i}$ satisfies the lemma. $\square$

Lemma 3. *For $1 \leq i \leq n$, let z_i be the rightmost vertex of τ_i, then $\mathcal{S}_{z_i}$ is a maximum weight independent set in $G[v_1, \ldots, v_i]$.*

Proof. The proof is again by induction on i. For $i = 1$ the lemma is obvious.

Suppose the claim holds for all $j \in \{1, \ldots, i-1\}$. Consider z_{i-1}, the rightmost vertex of τ_{i-1}, and let $\tilde{w}(z_{i-1})$ be its corresponding updated weight. Because insertion to τ_{i-1} is done rightmost in non-decreasing order:

$$\tilde{w}(z_{i-1}) \geq \tilde{w}(a) \ , \forall a \in \{v_1, \ldots, v_{i-1}\}.$$

By the induction hypothesis, $\mathcal{S}_{z_{i-1}}$ is a maximum weight independent set in $G[v_1, \ldots, v_{i-1}]$. When processing v_i, we scan τ_{i-1} from right to left to insert v_i and maintain the non-decreasing order of τ_{i-1}. Either $\tilde{w}(v_i) \geq \tilde{w}(z_{i-1})$, in which case, v_i is the rightmost vertex of τ_i, and hence $z_i = v_i$ and $\mathcal{S}_{v_i}$ is a maximum weight independent set in $G[v_1, \ldots, v_i]$, or $\tilde{w}(v_i) < \tilde{w}(z_{i-1})$ and so $z_i = z_{i-1}$ and $\mathcal{S}_{z_i} = \mathcal{S}_{z_{i-1}}$ remains a maximum weight independent set in $G[v_1, \ldots, v_i]$. $\square$

Theorem 1. *Algorithm 1 computes a maximum weight independent set of G when G is a cocomparability graph.*

Proof. This follows directly from Lemma 3. $\square$

To show that Algorithm 1 has complexity $\mathcal{O}(m+n)$ we have to explain some implementation details. We assume we are given an adjacency list representation of $G(V, E)$. Using the algorithm in [10], we compute a cocomparability ordering σ of the vertices of G, i.e., step 1 of Algorithm 1 is computed in $\mathcal{O}(m+n)$ time. The ordering $\sigma = \{v_1, v_2, \ldots, v_n\}$ is implemented using a doubly linked list.

In the remainder of the analysis, we denote by u the rightmost non-neighbour in τ_{i-1} of vertex v_i, if such a vertex u exists. In order to determine u, we create an array A of size n initialized to $A[k] = 0, \forall\ 1 \leq k \leq n$. At iteration i we update A such that $A[j] = i$ if and only if $v_j v_i \in E$; i.e. we keep the i^{th} row of the adjacency matrix of G, where $A[j] = i$ stands for $v_i v_j \in E$ and $A[j] < i$ for $v_i v_j \notin E$. Now for determining u it suffices to scan τ_{i-1} from right to left looking for the first vertex $v_j \in \tau_{i-1}$ such that $A[j] \neq i$; this vertex is then chosen to be u and we create a pointer p for vertex v_i that points to this rightmost non-neighbour u (this pointer is necessary to output the maximum independent set at the end). Once the weight of v_i is updated to $\tilde{w}(v_i) = \tilde{w}(u) + w(v_i)$, we scan τ_{i-1} from right to left once again to insert v_i into τ_{i-1}. To this end, we update the doubly linked lists pointers appropriately to maintain the increasing order of the weights in the new ordering. Since the pointer p keeps track of vertex u, we thus only need to scan the linked list of τ_{i-1} up to pointer p.

Now we can study the complexity of this algorithm. For every v_i, we first scan its adjacency list to update A, then we scan τ_{i-1} from right to left to determine u, and finally scan τ_{i-1} a second time to insert v_i and maintain the non-decreasing order of the updated weights. Setting the array A requires scanning v_i's adjacency list in any order and for every v_h in v_i's list we set $A[h] = i$. This operation takes $\mathcal{O}(d_{v_i})$ steps where d_{v_i} denotes the degree of v_i.

Scanning τ_{i-1} from right to left to determine u requires at most $\mathcal{O}(d_{v_i})$ checks to see whether $A[j] < i$ for $v_j \in \tau_{i-1}$. If such a vertex u exists then in constant time we update $\tilde{w}(v_i)$; similarly, in constant time we create the above mentioned pointer p that points to u. Otherwise, if u does not exist, v_i must be universal to all vertices in τ_{i-1}, and again it will cost at most $\mathcal{O}(d_{v_i})$ checks to conclude that no such u exists. Consequently, step 7 of Algorithm 1 takes $\mathcal{O}(d_{v_i})$ time per vertex.

Finally we need to insert v_i into τ_{i-1} to create τ_i. Since the weights of all vertices are strictly positive, we have $\tilde{w}(v_i) = w(v_i) + \tilde{w}(u) > \tilde{w}(u)$ and thus $u \prec_{\tau_i} v_i$. Since it takes at most $\mathcal{O}(d_{v_i})$ steps to determine u and $\tilde{w}(u) < \tilde{w}(v_i)$, it takes at most $\mathcal{O}(d_{v_i})$ comparisons to insert v_i into τ_{i-1} when scanning τ_{i-1} from right to left. Thus step 11 of Algorithm 1 takes at most $\mathcal{O}(d_{v_i})$ steps per vertex as well.

Step 13 can easily be determined in constant time with the use of a right-hand end pointer of τ_{i-1}. Thus all operations take at most $\mathcal{O}(d_{v_i})$ time per vertex; consequently when all vertices are processed, the for-loop in steps 6 to 12 of Algorithm 1 takes at most $\mathcal{O}(m+n)$ time in total. As already mentioned, step 1 is done in linear time [10], and clearly the for-loop in steps 2 to 5 takes linear too. Creating $\mathcal{S}_z$ in line 14 also takes linear time as it suffices to start at τ_n's righthand end pointer to find z and then unravel the p pointers starting with z's p pointer. We therefore conclude with the following theorem.

Theorem 2. *If $G(V, E, w)$ is a weighted cocomparability graph, then the maximum weight independent set of G can be computed in $\mathcal{O}(m+n)$ time.*

Robustness. To make the algorithm robust, it suffices to scan the neighbourhood of the vertex being processed to check if one of its neighbours appears earlier in the sub-solution.

More precisely, let v_i be the vertex being processed, and let v_j be the right most non-neighbour of v_i in τ_{i-1}. The algorithm would create $\mathcal{S}_i$ as $\mathcal{S}_j \cup \{v_i\}$. Before creating $\mathcal{S}_i$, we scan $N(v_i)$ to see if for some $x \in \mathcal{S}_j$, $xv_i \in E$. This operation takes $\mathcal{O}(d_{v_i})$. If such an x exists, the algorithm breaks and returns x, v_j, v_i as an umbrella. As v_i is the left most such vertex in σ, it follows that $xv_j, v_jv_i \notin E, xv_i \in E$. Since vertices are processed in the σ ordering, this umbrella occurs in σ, thus contradicting σ being a cocomparability order.

Suppose the algorithm returns a solution $\mathcal{S}$ even though the graph is not a cocomparability graph. Then it can be shown that $\mathcal{S}$ is still a maximum weight independent set. For this let $A = \{a_1, a_2, \ldots, a_k\}$ be an optimal solution of G such that $a_1 \prec_\sigma a_2 \prec_\sigma \ldots \prec_\sigma a_k$. Note that Lemma 2 still holds for G; for otherwise let v_i be the first vertex to break Lemma 2. Suppose $\mathcal{S}_i = \mathcal{S}_j \cup \{v_i\}$ is not a maximum weight independent set that v_i belongs to in $G[v_1, \ldots, v_i]$: Either (i) there exists a set $\mathcal{S}_h$ found by the algorithm for some vertex v_h such that $\mathcal{S}_h \cup \{v_i\}$ has a bigger weight than $\mathcal{S}_i$, or (ii) there exists a set of vertices $B = \{b_1, b_2, \ldots, b_t\}$ that was not created by the algorithm such that $B \subseteq \{v_1, \ldots, v_{i-1}\}$ and $B \cup \{v_i\}$ has a bigger weight than $\mathcal{S}_i$.

(i) Suppose such an $\mathcal{S}_h$ exists; then $v_j \prec_{\tau_{i-1}} v_h$ and $\mathcal{S}_h$ would have been chosen by the algorithm.

(ii) Suppose that the set B exists as defined above and let $b_1 \prec_\sigma b_2 \prec_\sigma \ldots \prec_\sigma b_t$ be the ordering of the elements of B as processed by the algorithm, i.e., as ordered by σ. By the choice of v_i, b_t satisfies Lemma 2, and thus b_t belongs to a set, call it $\mathcal{S}_\ell$, where $b_t = v_\ell$ for some $\ell \in \{1, \ldots, n\}$, such that

$\tilde{w}(v_\ell) \geq \sum_{s=1}^{t} w(b_s)$. By the choice of $\mathcal{S}_j$, we know that $\tilde{w}(v_j) \geq \tilde{w}(v_\ell) \geq \sum_{s=1}^{t} w(b_s)$, thus $\mathcal{S}_i = \mathcal{S}_j \cup \{v_i\}$ still satisfies Lemma 2.

Therefore when processing the elements of A as ordered by σ, there exists a set $\mathcal{S}_f$ (where $v_f = a_k$) that is a maximum weight independent set in $G[v_1, \ldots, v_f = a_k]$ containing a_k. Consequently $\mathcal{S}_f$ is an optimal solution, and since the algorithm does not break, $\mathcal{S}_f$ remains an optimal solution throughout all iterations and is returned by the algorithm. The same argument holds if there exists more than one optimal solution. Therefore the algorithm is robust as it either returns an umbrella (showing that the given ordering was not a cocomparability order) or an optimal solution.

Minimum Vertex Cover. It is a well known fact that for a maximum independent set S of a graph G, the set $V \backslash S$ is a minimum vertex cover of G. If we chose S to be the independent set returned by Algorithm 1, we therefore get the following corollary.

Corollary 1. *If $G(V, E, w)$ is a weighted cocomparability graph, with weight function $w : V \to \mathbb{R}_{>0}$, a minimum weight vertex cover of G can be computed in $\mathcal{O}(m+n)$ time.*

Acknowledgements

The second author wishes to thank University of Toronto, Department of Computer Science, and the Natural Science and Engineering Research Council of Canada for their financial support, through grants number RGPIN 7631-12 and RGPIN 7671-11. Furthermore, the authors wish to thank their anonymous referees for suggesting certification which lead us to the robustness of the algorithm.

References

[1] M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Annals of Discrete Mathematics, Vol 57, North-Holland Publishing Co., Amsterdam, The Netherlands, 2004.

[2] E. M. Arkin, E. B. Silverberg, Scheduling jobs with fixed start and end times, Discrete Appl. Math. 18 (1) (1987) 1–8.

[3] S. De Vries, R. V. Vohra, Combinatorial auctions: A survey, INFORMS Journal on computing 15 (3) (2003) 284–309.

[4] V. Bafna, B. Narayanan, R. Ravi, Nonoverlapping local alignments (weighted independent sets of axis-parallel rectangles), Discrete Appl. Math. 71 (1) (1996) 41–53.

[5] D. Kratsch, L. Stewart, Domination on cocomparability graphs, SIAM J. Disc. Math. 6 (3) (1993) 400–417.

[6] Y. Daniel Liang, M.-S. Chang, Minimum feedback vertex sets in cocomparability graphs and convex bipartite graphs, Acta Informatica 34 (5) (1997) 337–346.

[7] G. B. Mertzios, D. G. Corneil, A simple polynomial algorithm for the longest path problem on cocomparability graphs, SIAM J. Disc. Math. 26 (3) (2012) 940–963.

[8] D. G. Corneil, B. Dalton, M. Habib, LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs, SIAM J. Comput. 42 (3) (2013) 792–807.

[9] E. Dahlhaus, J. Gustedt, R. M. McConnell, Partially complemented representations of digraphs., Discrete Mathematics & Theoretical Computer Science 5 (1) (2002) 147–168.

[10] R. M. McConnell, J. P. Spinrad, Modular decomposition and transitive orientation, Discrete Math. 201 (1) (1999) 189–241.