Minimal comparability completions of arbitrary graphs*

Pinar Heggernes† Federico Mancini† Charis Papadopoulos†

Abstract

A transitive orientation of an undirected graph is an assignment of directions to its edges so that these directed edges represent a transitive relation between the vertices of the graph. Not every graph has a transitive orientation, but every graph can be turned into a graph that has a transitive orientation, by adding edges. We study the problem of adding an inclusion minimal set of edges to an arbitrary graph so that the resulting graph is transitively orientable. We show that this problem can be solved in polynomial time, and we give a surprisingly simple algorithm for it. We use a vertex incremental approach in this algorithm, and we also give a more general result that describes graph classes Π for which Π completion of arbitrary graphs can be achieved through such a vertex incremental approach.

1 Introduction

A transitive orientation of an undirected graph is an assignment of a direction to each of the edges, such that the edges represent a binary transitive relation on the vertices. An undirected graph is a *comparability graph* if there is a transitive orientation of its edges, and hence comparability graphs are also called *transitively orientable graphs*. This is a wide and well known graph class studied by many authors, and and it has applications in areas like archeology, psychology, and political sciences [2, 12]. Comparability graphs are perfect, and they can be recognized in polynomial time. Many interesting optimization problems that are NP-hard on arbitrary graphs, like coloring and maximum (weighted) clique, are polynomially solvable on comparability graphs [2]. Hence, computing a comparability supergraph of an arbitrary graph, and solving a generally NP-hard problem in polynomial time on this supergraph, is a way of obtaining approximation algorithms for several hard problems. For graphs coming from the application areas mentioned above, there may be missing edges due to lacking data so that the graph fails to be comparability, in which case one is again interested in computing a comparability supergraph. A comparability graph obtained by adding edges to an arbitrary graph is called a *comparability completion* of the input graph. Unfortunately, computing a comparability completion with the minimum number of added edges (called a *minimum* completion) is an NP-hard problem [3].

A *minimal* comparability completion H of G is a comparability completion of G such that no proper subgraph of H is a comparability completion of G. Although the number of added edges in a minimal comparability completion may be far from minimum, computing a few different minimal comparability completions, and choosing the one with the smallest number of edges is a possible approach to finding a comparability completion close to minimum. Furthermore, the

*This work is supported by the Research Council of Norway through grant 166429/V30.

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Emails: {pinar, federico, charis}@ii.uib.no

set of minimal comparability completions of a graph contains the set of minimum comparability completions. Therefore, the study of minimal comparability completions is a first step in the search for minimum comparability completions, possibly through methods like exact exponential time algorithms or parameterized algorithms. In this paper, we give the first polynomial time algorithm for computing minimal comparability completions of arbitrary graphs, and hence we show that this problem is solvable in polynomial time, as opposed to computing minimum comparability completions.

The study of minimal completions of arbitrary graphs into a given graph class started with a polynomial-time algorithm for minimal chordal completions in 1976 [13], before it was known that minimum chordal completions are NP-hard to compute [15]. Since then the NP-hardness of minimum completions has been established for several graph classes (summarized in [10]). Recently, several new results, some of which have been presented at recent years' SODA and ESA conferences, have been published on completion problems, leading to faster algorithms for minimal chordal completions [6, 8, 9], and polynomial-time algorithms for minimal completions into split, interval, and proper-interval graphs [4, 5, 11]. The complexity of computing minimal comparability completions has been open until now.

There are simple examples to show that a minimal comparability completion cannot be obtained by starting from an arbitrary comparability completion, and removing unnecessary edges one by one (as opposed to minimal completions into chordal and split graphs). To overcome this difficulty, we use a vertex incremental approach in our algorithm. A vertex incremental algorithm has also proved useful for minimal completions into interval graphs [5], and therefore we find it worthwhile to give a more general result here, describing classes of graphs into which minimal completions of arbitrary graphs can be computed with such a vertex incremental approach. Notice, however, that the algorithm for each step is completely different for, and dependent on, each graph class, and polynomial time computability is not guaranteed by the vertex incremental approach.

This paper is organized as follows. In the next section we give some notation and background on comparability graphs and a new result on vertex incremental minimal completions. In Section 3 we present an algorithm for the vertex incremental step: Given a comparability graph G (which is the minimal comparability completion of the previous incremental step) and a new vertex x which is added to G along with a given set of edges between x and G, compute a minimal comparability completion of this augmented graph. $G_x = (V(G) \cup \{x\},\ E(G) \cup \{xv \mid v \in N_x\}\})$. We prove the correctness of the given algorithm in Section 4, and discuss the time complexity issues in Section 5. We conclude in Section 6.

2 Notation and background

We consider undirected finite graphs with no loops or multiple edges. For a graph G, we denote its vertex and edge set by $V(G)$ and $E(G)$, respectively, with $n = |V(G)|$ and $m = |E(G)|$. For a vertex subset $S \subseteq V(G)$, the subgraph of G induced by S is denoted by $G[S]$. Moreover, we denote by $G - S$ the graph $G[V(G) - S]$ and by $G - v$ the graph $G[V(G) - \{v\}]$.

The *neighborhood* $N_G(x)$ of a vertex x of the graph G is the set of all the vertices of G which are adjacent to x. The *closed neighborhood* of x is defined as $N_G[x] = N_G(x) \cup \{x\}$. If $S \subseteq V(G)$, then the neighbors of S, denoted by $N_G(S)$, are given by $\left(\bigcup_{x \in S} N_G(x)\right) - S$. For a vertex x of G, the set $N_G(N_G(x)) - \{x\}$ is denoted by $N_G^2(x)$. For a pair of vertices x, y of a graph G we

call xy a *non-edge* of G if $xy \notin E(G)$. A vertex x of G is *universal* if $N_G[x] = V(G)$.

A *clique* is a set of pairwise adjacent vertices while an *independent set* is a set of pairwise non-adjacent vertices. A graph is *bipartite* if its vertex set can be partitioned into two independent sets. Bipartite graphs are exactly the class of graphs that do not contain cycles of odd length.

Given a new vertex $x \notin V(G)$ and a set of vertices N_x of G, we denote by G_x the graph obtained by adding x to G and making x adjacent to each vertex in N_x, i.e., $V(G_x) = V(G) \cup \{x\}$ and $E(G_x) = E(G) \cup \{xv \mid v \in N_x\}$; thus $N_{G_x}(x) = N_x$. For a vertex $x \notin V(G)$, we denote by $G + x$ the graph obtained by adding an edge between x and *every* vertex of $V(G)$, thus x is universal in $G + x$.

2.1 Comparability graphs

A *digraph* is a directed graph, and an *arc* is a directed edge. While we denote an undirected edge between vertices a and b equivalent by ab or ba, we denote an arc from a to b by (a, b), and an arc in the opposite direction by (b, a). A directed acyclic graph (*dag*) is *transitive* if, whenever (a, b) and (b, c) are arcs of the dag, (a, c) is also an arc. An undirected graph is a *comparability* graph if directions can be assigned to its edges so that the resulting digraph is a transitive dag, in which case this assignment is called a *transitive orientation*.

We consider an undirected graph G to be a symmetric digraph, that is, if $xy \in E(G)$ then (x, y) and (y, x) are arcs of G. Two arcs (a, b) and (b, c) of an undirected graph G are called *incompatible* if ac is not an edge of G. We say, then, that (a, b) is incompatible with (b, c) and vice versa, or that $((a, b), (b, c))$ is an incompatible pair. The *incompatibility graph* B_G of an undirected graph G is defined as follows: In B_G there is one vertex for each arc of G, and therefore we will (somewhat abusively) denote a vertex of B_G that corresponds to arc (a, b) of G by (a, b). For each edge ab of G, there are two adjacent vertices (a, b) and (b, a) in B_G. In addition, there is an edge between two vertices (a, b) and (b, c) of B_G if and only if arcs (a, b) and (b, c) are incompatible in G. We will refer to the edges of B_G of this latter type as *incompatibilities*. Since we consider an undirected graph to be a symmetric digraph, if $(a, b)(b, c)$ is an edge (incompatibility) of B_G then $(c, b)(b, a)$ is also an edge (incompatibility) of B_G. An example of a graph G and its incompatibility graph B_G is given in Figure 1.

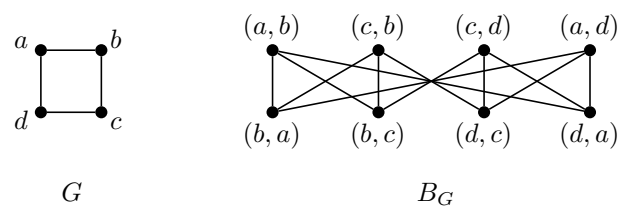

Figure 1: *A graph G and its incompatibility graph B_G.*

The incompatibility graph will be our main tool to compute minimal comparability completions, and the following result from Kratsch et al. [7] is central to our algorithm.

Theorem 1 ([7]). *An undirected graph G is a comparability graph if and only if its incompatibility graph B_G is bipartite.*

It is mentioned in [7] that a transitive orientation of a comparability graph G must be an independent set of the bipartite graph B_G. Note that for every edge ab of G, exactly one of

the vertices (a, b) and (b, a) of B_G are in the same independent set. So by choosing one of the independent set of B_G, we choose an orientation of the edges of G. For the example in Figure 1, we see that a good (transitive) orientation is $(a, b), (c, b), (c, d), (a, d)$.

2.2 A vertex incremental approach for minimal completions

A comparability graph can be obtained from any graph G by adding edges, and the resulting graph is called a *comparability completion* of G. An edge that is added to G to obtain a comparability completion H is called a *fill edge*. A comparability completion $H = (V, E \cup F)$ of $G = (V, E)$, with $E \cap F = \emptyset$, is *minimal* if $(V, E \cup F')$ fails to be a comparability graph for every $F' \subset F$. We will now show that minimal comparability completions can be obtained vertex incrementally. It was shown previously that minimal triangulations [1] and minimal interval completions [5] can be computed incrementally. Therefore, we give a more general result here, describing graph classes into which minimal completions of arbitrary graphs can be computed by a vertex incremental approach. Let Π be a graph class. Speaking about Π completions (defined analogously to comparability completions) of arbitrary graphs is only meaningful if every graph can be embedded in a graph of Π by adding edges. For example, if complete graphs belong to Π then any graph has a Π completion. A graph class Π is called *hereditary* if all induced subgraphs of graphs in Π also belong to Π.

Property 2. *We will say that a graph class Π has the* universal vertex property *if, for every graph $G \in \Pi$ and a vertex $x \notin V(G)$, $G + x \in \Pi$.*

Lemma 3. *Let H be a minimal Π completion of an arbitrary graph G, and let G_x be a graph obtained from G by adding a new vertex x adjacent to some vertices of G. If Π is hereditary and has the universal vertex property, then there is a minimal Π completion H' of G_x such that $H' - x = H$.*

Proof. Let H_x be the graph obtained by adding x to H together with the edges between x and $N_{G_x}(x)$. Observe first that a Π completion of H_x can be obtained by adding edges only incident to x, since $H + x \in \Pi$. Thus, a minimal Π completion of H_x can be obtained by adding a subset of the edges between x and $V(H_x) - N_{G_x}(x)$. Let H' be a minimal Π completion of H_x obtained by adding edges incident to x. Obviously, $H' - x = H$. Assume for the sake of contradiction that H' is not a minimal Π completion of G_x. This means that a subset of the newly added edges to H_x to obtain H' and a nonempty subset of the edges added to G to obtain H can be removed from H' without destroying the Π property. But since Π is hereditary, this contradicts that H is a minimal Π completion of G. Thus H' must be a minimal Π completion of G_x. $\square$

An important consequence of Lemma 3 is that for a hereditary graph class Π with the universal vertex property, a minimal Π completion of any input graph G can be computed by introducing the vertices of G in an arbitrary order $x_1, x_2, \ldots, x_n$. Given a minimal Π completion H_i of $G_i = G[x_1, \ldots, x_i]$, we compute a minimal Π completion of $G_{i+1} = G[x_1, \ldots, x_i, x_{i+1}]$ by actually computing a minimal Π completion of the graph $H_{x_{i+1}} = (\{x_1, \ldots, x_{i+1}\},\ E(H_i) \cup \{x_{i+1}v \mid v \in N_{G_{i+1}}(x_{i+1})\})$. In this completion, we add *only* fill edges incident to x_{i+1}. Meanwhile, notice that this minimal completion is not necessarily easy to obtain, and some major challenges might need to be overcome, depending on the graph class Π. Note also that all minimal completions of G cannot be created in this way, since by allowing only addition of fill edges incident to the incremental vertex x, we rule out several possible minimal completions.

Observation 4. *The class of comparability graphs is hereditary and satisfies the universal vertex property.*

Proof. The transitive orientation property is clearly hereditary (see for example [2]). Let G be a comparability graph and $x \notin V(G)$. We will show that $G + x$ is a comparability graph. We know that G has a transitive orientation D of its edges. Let us give the following orientation to the edges of $G+x$: For edges of G, we orient them as in D. For edges incident to x, we orient all of them towards x. Now the pairs of arcs of this digraph that can cause a problem are all of type $((a, b), (b, x))$. But since x is universal, ax is also an edge of $G + x$, and it is oriented towards x. Thus the described orientation is transitive on $G + x$, and therefore $G + x$ is comparability. □

The real challenge is how to do the computations of each vertex incremental step. This is exactly the problem that we solve in the rest of this paper. Thus for the rest of the paper, due to Lemma 3 and Observation 4, we consider as input a comparability graph G and a new vertex $x \notin V(G)$ together with a list of vertices N_x in G. Our aim is to compute a minimal comparability completion of $G_x = (V(G) \cup \{x\},\ E(G) \cup \{xv \mid v \in N_x\})$. We do this by finding an appropriate set of fill edges F_x incident to x such that we obtain a comparability graph by adding F_x to G_x, and no proper subset F_x yields a comparability graph when added to G_x.

3 An algorithm for minimal comparability completion of G_x

In this section, we give an algorithm that computes a minimal comparability completion H of G_x, for a given comparability graph G and a new vertex $x \notin V(G)$ together with a neighborhood N_x in G. Our main tool will be the incompatibility graph B_G of G, which we know is bipartite by Theorem 1. We will proceed to update B_G with the aim of obtaining the incompatibility graph B_{G_x} of G_x. We will keep this partial incompatibility graph a bipartite graph at each step. If G_x is not a comparability graph, we will have to add fill edges to G_x to be able to achieve this goal.

Let $E_x = \{xv \mid v \in N_x\}$ (thus $G_x = (V \cup \{x\}, E \cup E_x)$). Our first step in obtaining B_{G_x} from B_G is to add vertices corresponding to edges of E_x and the edges and incompatibilities between these. We will make a separate graph B_x to represent the incompatibilities among the edges of E_x. Let B_x be the graph that has two adjacent vertices (x, v) and (v, x) for each $xv \in E_x$, and that has all incompatibilities that are implied by non-edges of G_x between vertices of N_x. To be more precise, if $\mathcal{E} = \{(x, v) \mid xv \in E_x\} \cup \{(v, x) \mid xv \in E_x\}$, and $B_{G_x[N_x \cup \{x\}]}$ is the incompatibility graph of $G_x[N_x \cup \{x\}]$, then B_x is the subgraph of $B_{G_x[N_x \cup \{x\}]}$ induced by $\mathcal{E}$. An example is given in Figure 2. Observe that the graph $G_x[N_x \cup \{x\}]$ is a comparability graph, since $G[N_x]$ is comparability by the hereditary property, and x is a universal vertex in $G_x[N_x \cup \{x\}]$. Following the above arguments, B_x is a bipartite graph by Theorem 1.

For our purposes, we also need to define the set of incompatibilities of B_G implied by a given non-edge xv of G. We call this set $C_G(xv)$, and define it as follows for each non-edge xv of G.

$$C_G(xv) = \{(x, w)(w, v) \mid w \in N_G(x) \cap N_G(v)\} \cup \{(v, w)(w, x) \mid w \in N_G(x) \cap N_G(v)\}.$$

Observe that $C_G(e_1) \cap C_G(e_2) = \emptyset$ for any pair of non-edges e_1 and e_2 of G, and $\bigcup_{e \notin E(G)} C_G(e)$ is exactly the set of all incompatibilities in B_G.

Lemma 5. *By adding the set of edges $C_{G_x}(xv)$ for each $v \in N^2_{G_x}(x)$ into the graph $B_G \cup B_x$, we obtain the incompatibility graph B_{G_x} of G_x.*

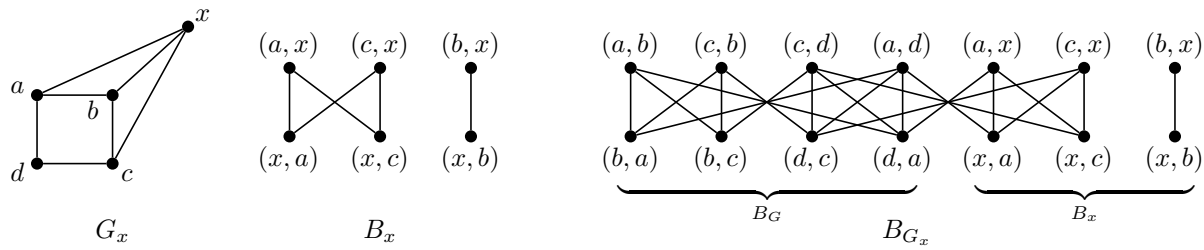

Figure 2: An example that shows G_x, B_x, and B_{G_x}, for the graph G given in Figure 1.

Proof. Adding a new vertex x to G and some edges incident to x, can only create incompatibilities between pairs of arcs that both have an endpoint in x and between pairs of arcs where one has an endpoint in x and the other has an endpoint in $N^2_{G_x}(x)$. The incompatibilities of the first type are already present as edges in B_x. The incompatibilities of the second type are exactly the ones given by $C_{G_x}(xv)$. Notice that the graph $B_G \cup B_x$ does not contain any edges between vertices of B_G and B_x. By definition, all vertices of B_{G_x} (arcs of edges in G_x) are contained in $B_G \cup B_x$, and $B_G \cup B_x$ has no further vertices. Hence the result follows. □

Assume that we want to compute the incompatibility graph B_{G_x} of G_x. We start with the partial incompatibility graph $B_G \cup B_x$, which is bipartite by the above arguments. By Lemma 5, to get B_{G_x} it is sufficient to scan all non-edges of G_x between x and $N^2_{G_x}(x)$ one by one, and add the incompatibilities that are implied by each non-edge into the partial incompatibility graph. If G_x is a comparability graph, then by Theorem 1, the partial incompatibility graph will stay bipartite at each step, since we never delete edges from it. By the same argument, if G_x is not a comparability graph, then at some step, when we add the incompatibilities implied by a non-edge, we will get an odd cycle in the partial incompatibility graph. For computing a minimal comparability completion H of G_x, we augment this approach as follows: If adding the incompatibilities implied by non-edge xv results in a non-bipartite partial incompatibility graph, then we do not add these incompatibilities, and instead, we decide that xv should become a fill edge of H. Note that, by starting with the partial bipartite graph $B_G \cup B_x$, we force all possible fill edges to be incident to x; all the incompatibilities of some non-edge xv can be removed by the addition of fill edge xv to H.

At start, we let $L = \{xv \mid v \in N^2_{G_x}(x)\}$, $B = B_G \cup B_x$, and $H = G_x$. For each non-edge $xv \in L$, we check whether or not non-edge xv should become a fill edge of the intermediate graph H, using the information given by $C_H(xv)$ and B. If $B \cup C_H(xv)$ is a bipartite graph, then we update $B = B \cup C_H(xv)$ and decide that xv *will never become a fill edge.* In the opposite case, we add *fill edge* xv to H, and update B as follows.

1. Add the two adjacent vertices (x, v) and (v, x) in B.

2. For each new incompatible pair $((z, x), (x, v))$ or $((v, x), (x, z))$ in H, add the corresponding edge (incompatibility) to B connecting the vertices of the pair. (We will show that this can never introduce odd cycles in the incompatibility graph.)

3. For each new incompatible pair $((x, v), (v, u))$ or $((u, v), (v, x))$ in H, add the corresponding edge (incompatibility) to B connecting the vertices of the pair *only if xu is a non-edge that has already been processed and decided to stay a non-edge (marked).* If not, either $xu \in L$ or we add it to L.

The second case takes care of new incompatibilities among the edges incident to x, and the last case takes care of all other new incompatibilities. In the last case, when we encounter new incompatibilities that are implied by a non-edge e which we have not yet processed, we do not add these incompatibilities to B at this step, and we wait until we come to the step which processes e. The reason for this is the following: If we add these incompatibilities now, and later decide that e should become a fill edge, then we have to delete these incompatibilities from B. This causes problems regarding minimality, because deleting "old" incompatibilities can make some previously added fill edges become redundant, and thus we might have to examine each initial non-edge several times. When we do not add the incompatibilities before they are needed, we never have to delete anything from B, and B can only grow at each step. This way, the intermediate graph B will at all steps be a supergraph of $B_G \cup B_x$ and a subgraph of B_H. This is the clue to the simplicity of our algorithm, which makes it sufficient to examine each non-edge incident to x *once*.

The non-edges that are removed from L are marked, which means that they will stay non-edges. This marking is necessary since new non-edges enter L during the algorithm, and we need to test for every incompatibility we discover, whether it is already implied by a marked non-edge so that we can add it at this step, or we should wait. The details of the algorithm called *Minimal_Comparability_Completion* (MCC) are given below.

```
Algorithm: Minimal_Comparability_Completion (MCC)
Input: A comparability graph G, B_G, and G_x for a vertex x ∉ V(G)
Output: A minimal comparability completion H of G_x, and B = B_H
B = B_G ∪ B_x;  L = {xv | v ∈ N^2_{G_x}(x)};  H = G_x;
Unmark all non-edges of H incident to x;
while L ≠ ∅ do
    Choose a non-edge xv ∈ L;
    if B ∪ C_H(xv) is a bipartite graph then
        B = B ∪ C_H(xv);
    else
        Add fill edge xv to H;
        Add vertices (x, v) and (v, x) and an edge between them to B;
        forall z ∈ N_H(x) and z ∉ N_H[v] do
            Add edges (v, x)(x, z) and (z, x)(x, v) to B;
        forall u ∈ N_H(v) and u ∉ N_H[x] do
            if xu is marked then
                Add edges (x, v)(v, u) and (u, v)(v, x) to B;
            else if xu ∉ L then
                Add xu to L;
    Mark xv and remove it from L;
```

4 Correctness of Algorithm MCC

Although our algorithm is surprisingly simple due to the fact that each non-edge is examined once, its proof of correctness is quite involved, and requires a series of observations and lemmas, some of which with long proofs. Let us define a *step* of the algorithm to be one iteration of the while–loop given between lines 3–17. For the proof of correctness, we will sometimes need to distinguish between the graph H at the start of a step and the updated graph H at the end of a step, to consider the changes made at one step. Throughout the rest of the paper, let H_I be the graph H at the start of step I, and let H_{I+1} be the graph obtained at the end of this step, and define B_I and B_{I+1} analogously.[1]

Observation 6. *Let I be the step of the algorithm that processes the non-edge $xv \in L$. Then B_I contains no edge belonging to $C_{H_I}(xv)$.*

Proof. Assume for the sake of contradiction that B_I contains an edge $(x, w)(w, v)$ belonging to $C_{H_I}(xv)$. This can happen only if there is a vertex $w \in N_{H_I}(x) \cap N_{H_I}(v)$ such that xw is a fill edge of H_I. But by line 13 of the algorithm the incompatibility $(x, w)(w, v)$ cannot have been added previously, since, being processed for the first time, xv is unmarked at all previous steps. Thus no edge of $C_{H_I}(xv)$ is contained in B_I. □

Lemma 7. *At the end of each step of the algorithm, B_I is a subgraph of the incompatibility graph B_{H_I} of H_I.*

Proof. We prove this by induction on the number of steps. At start, $B = B_G \cup B_x$ is definitely a subgraph $B_1 = B_{G_x}$. Consider any step I of the algorithm. By the induction hypothesis, we can assume that B_I is a subgraph of B_{H_I}, and we must show that B_{I+1} is a subgraph of $B_{H_{I+1}}$.

Let xv be the non-edge of L that we process at step I. If $B_I \cup C_{H_I}(xv)$ is bipartite then no fill edge is added at this step and we have $H_{I+1} = H_I$ and thus $B_{H_{I+1}} = B_{H_I}$. Note also that $C_{H_I}(xv)$ is a subset of the edges of $B_{H_{I+1}}$ by definition. Hence, in this case the graph $B_{I+1} = B_I \cup C_{H_I}(xv)$ is a subgraph of $B_{H_{I+1}}$.

In case the graph $B_I \cup C_{H_I}(xv)$ is not bipartite, B_{I+1} is obtained from B_I by adding two adjacent vertices (x, v) and (v, x) and the corresponding incompatibilities induced by the addition of the edge xv into H_I. These new edges correspond to incompatible pairs of H_{I+1} of the form $((x, v), (v, u))$ or $((u, v), (v, x))$, and of the form $((z, x), (x, v))$ or $((v, x), (x, z))$. By definition, the graph $B_{H_{I+1}}$ contains this kind of edges of B_{I+1}. We see that all edges added to B_I to obtain B_{I+1} are also edges of $B_{H_{I+1}}$. Hence the only way B_{I+1} can fail to be a subgraph of $B_{H_{I+1}}$ is if B_I has edges that do not belong to $B_{H_{I+1}}$. Assume that there is an incompatibility p in B_I which should not be present in $B_{H_{I+1}}$. This can happen only if the addition of fill edge xv removes this incompatibility p at step I. This means that p is an incompatibility implied by the non-edge xv and thus p belongs to $C_{H_I}(xv)$. But by Observation 6, B_I contains no edge of $C_{H_I}(xv)$, thus this situation cannot happen, and B_{I+1} is a subgraph of $B_{H_{I+1}}$. □

We have thus proved that B_I is at all times a partial incompatibility graph of the intermediate graph H_I. At the end of the algorithm, since all non-edges that can cause incompatibilities are scanned, and all such incompatibilities are added, we will argue that B_I is indeed the correct incompatibility graph of H_I. What remains to prove is that B_I is a bipartite graph at all steps.

[1] Unconventionally, we need to use a capital letter as index, since all small letters as i, j, k, l are used in the proofs of the results of this section.

This is obvious if xv is not added as a fill edge at the step that processes xv, but it has to be shown in the case xv is added as a fill edge. First we introduce the notion of conflicts.

Definition 8. *At each step of the algorithm, a non-edge xv of the intermediate graph H_I is called a* conflict *if $B \cup C_{H_I}(xv)$ is not a bipartite graph.*

Lemma 9. *Let I be the step of the algorithm that processes non-edge $xv \in L$. If xv is a conflict then H_I is not a comparability graph.*

Proof. Follows from Lemma 7 and Theorem 1 since an odd cycle in B_I cannot disappear by the addition of edges or vertices in B_I. □

Now we start the series of results necessary to prove that at each step B_I is a bipartite graph. We will prove this by induction on the number of steps. For each step I, we will assume that B_I is bipartite, and show that this implies that B_{I+1} is bipartite. Since $B_1 = B_x \cup B_G$ is bipartite, the result will follow.

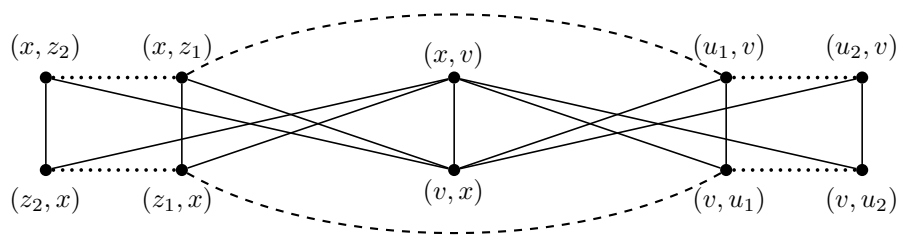

Figure 3: Adding the fill edge xv in B.

Let z_1, z_2 and u_1, u_2 be vertices of H_I which fulfill the conditions of the first for–loop and the second for–loop of Algorithm MCC, respectively. With the following result we establish the situations that occur in B_I whenever an odd cycle appears in B_{I+1} (see also Figure 3).

Observation 10. *Assume that B_I is bipartite. If xv is conflict at step I, then B_{I+1} is not bipartite only if there is a path on even number of vertices in B_I between the following pair of vertices: (i) $((x, z_1), (x, z_2))$ or (ii) $((v, u_1), (v, u_2))$ or (iii) $((x, z_1), (u_1, v))$.*

Proof. Given that B_I is bipartite, if B_{I+1} contains an odd cycle, it must contain at least one vertex (x, v) or (v, x), since we do not add edges between the other vertices. Considering the edge between (x, v) and (v, x) and the symmetry of their neighborhood, there are only two ways an odd cycle can be created: (x, v) (or (v, x)) plus an even path between two of its neighbors (notice that the three cases of the observation cover all possibilities); or (x, v) and (v, x) plus an odd path between a neighbor of (x, v) and a neighbor of (v, x). These two cases are equivalent, in fact if there is an odd path between a neighbor (a, b) of (x, v) and a neighbor (c, d) of (v, x), then there is an even path between the neighbors (a, b) and (d, c) of (x, v) by the symmetry of their neighborhood. Hence, the pointed cases describe the existence of an odd cycle in B_{I+1}. □

Our goal is to show that these cases cannot happen in B_I, and therefore B_{I+1} remains a bipartite graph. We prove each case by showing that if such a path exists then there is an odd cycle in B_I which is a contradiction to our assumption that B_I is a bipartite graph.

For the following results, we denote a cycle on k vertices by C_k and a path on k vertices by P_k. A path or a cycle is *even* or *odd* according to the parity of its number of vertices. Let G

be a graph and B_G be its incompatibility graph. We denote a path on $k-1$ vertices in B_G in the following form: $P = (x_1, x_2)(x_2, x_3)\dots(x_{k-1}, x_k)$; recall that a pair of adjacent vertices of G represents a vertex of B_G. By definition, if a path P in B_G connects the vertices (x_1, x_2) and (x_{k-1}, x_k) then there exists also the transposed path of P denoted by P^T which connects the vertices (x_k, x_{k-1}) and (x_2, x_1), i.e., $P^T = (x_k, x_{k-1})\dots(x_3, x_2)(x_2, x_1)$. Recall also that there is always an edge $(x, y)(y, x)$ in B_G for each edge xy in G.

Lemma 11. *If there is an even (respectively, odd) path connecting vertices (a, b) and (c, d) of B_G then it has the following form:*

$$P_{k+3} = (a, b)(b, q_1)(q_1, q_2)(q_2, q_3)\dots(q_{k-1}, q_k)(q_k, c)(c, d),$$

where k is an odd (respectively, even) number, $aq_1, bq_2, q_kd, cq_{k-1} \notin E(G)$, and $q_iq_{i+2} \notin E(G)$ for $1 \le i \le k-2$.

Proof. By the definition of the edges of the incompatibility graph B_G we have two types of edges among two vertices of B_G: either $(a, b)(b, c)$ or $(b, a)(c, b)$ such that $ab \notin E(G)$. The form of the path shown in the lemma uses only the first kind of edges. But any edge (path on two vertices) of the kind $(b, a)(c, b)$ can be turned into a path on four vertices using only the first form of edges: $(b, a)(a, b)(b, c)(c, b)$. Thus an even or odd path between two vertices of B_G has the form of the equation as shown and the constraints for the non-edges are justified by definition; otherwise there is no path connecting the vertices. □

Suppose that B_I is bipartite. If xv is a conflict at step I, then there is an inclusion maximal subset $C'_{H_I}(xv)$ of $C_{H_I}(xv)$ such that $B_I \cup \{C'_{H_I}(xv)\}$ is a bipartite graph. For the rest of this section we define $B'_I = B_I \cup \{C'_{H_I}(xv)\}$. Thus if B_I is bipartite, so is B'_I, and any of the incompatibilities of $C_{H_I}(xv) \setminus C'_{H_I}(xv)$ results in an odd cycle if added to B'_I. This is formalized in the following observation.

Observation 12. *Assume that B_I is bipartite. If xv is a conflict at step I, then there is a path on odd number of vertices in B'_I connecting (x, w) and (w, v), for some $w \in N_{H_I}(x) \cap N_{H_I}(v)$.*

Observation 13. *Assume that B_I is bipartite. If xv is a conflict at step I, then there is a path in B'_I of the form:*

$$P_{xv} = (x, w)(w, p_1)(p_1, p_2)\dots(p_{\ell-1}, p_\ell)(p_\ell, w)(w, v),$$

where ℓ is an even number, $x \neq p_\ell$ and $v \neq p_1$.

Proof. By Lemma 11, P_{xv} contains $\ell + 3$ vertices. Notice that by the definition of the odd path, $xp_1, p_{\ell-1}w, p_\ell v \notin E(H_I)$, and ℓ is an even number by the odd number of vertices in P_{xv}. Since xv is a conflict the incompatibility $(x, w)(w, v)$ is not present in B'_I and thus $x \neq p_\ell$ and $v \neq p_1$. □

Now let us show that the incompatibilities added during the first forall-loop starting at line 10 do not create any odd cycles.

Lemma 14. *Assume that B_I is bipartite. If xv is a conflict at step I then there is no path on even number of vertices connecting (x, z_1) and (x, z_2) in B'_I, for every pair of vertices z_1, z_2 such that $z_1, z_2 \in N_{H_I}(x)$ and $z_1, z_2 \notin N_{H_I}[v]$.*

Proof. Assume for the sake of contradiction that there is such an even path connecting them. Then by Lemma 11 it has the following form:

$$P_z = (x, z_1)(z_1, q_1)(q_1, q_2) \dots (q_{k-1}, q_k)(q_k, x)(x, z_2),$$

where $k \geq 3$ is an odd number. If $k = 1$ then there is no path on even number of vertices connecting (x, z_1) and (x, z_2). Observe that the path P_z contains $k + 3$ vertices. Notice that $xq_1, z_1q_2 \notin E(H_I)$ and $z_2q_k, xq_{k-1} \notin E(H_I)$ and $q_iq_{i+2} \notin E(H_I)$, for $1 \leq i \leq k - 2$; otherwise there is no even path (see also Lemma 11). Considering the path P_z in B'_I, we have to distinguish between when z_1 and z_2 are adjacent in H_I and when they are not. We will prove that in each case there is an odd cycle in B'_I which is a contradiction since B'_I is a bipartite graph.

- *Case 1: $z_1z_2 \notin E(H_I)$.*
  In this case it is easy to see that appending the pairs (z_2, x) and (x, z_1) in P_z we obtain an odd cycle in B'_I:
  $$C_{k+4} = P_z \underbrace{(z_2, x)(x, z_1)}_{z_1z_2 \notin E(H_I)}.$$

- *Case 2: $z_1z_2 \in E(H_I)$.*
  In this case we have to consider also the fact that xv is a conflict. By Observation 12 there is a vertex w which induces an odd path P_{xv} in B'_I. We distinguish between the cases where w is (i) non-adjacent to both z_1, z_2, (ii) adjacent only to one of them and (iii) adjacent to both of them.

  - *Case 2.1: $wz_1 \notin E(H_I)$ and $wz_2 \notin E(H_I)$.*
    In this case it is easy to see that the following odd cycle occurs in B'_I:
    $$C_{k+6} = (w, x)P_z \underbrace{(z_2, x)(x, w)}_{wz_2 \notin E(H_I)}(w, x).$$

  - *Case 2.2: $wz_1 \notin E(H_I)$ and $wz_2 \in E(H_I)$.*
    By Observation 13 there is an odd path P_{xv} connecting (x, w) and (w, v) in B'_I; recall that the path P_{xv} contains $\ell+3$ vertices, where ℓ is an even number and $xp_1 \notin E(H_I)$. Here we prove that if there is an even path which connects (x, z_1) and (x, z_2) then there is a path $P_{z_2z_1}$ on r vertices where r is an even number which connects (z_2, z_1) and (z_1, x). Hence the result follows based on the path $P_{z_2z_1}$, since the following odd cycle appears in B'_I:
    $$C_{\ell+r+5} = (w, z_2)P_{z_2z_1}P_{xv}(v, w)(w, z_2).$$
    In order to prove the existence of the path $P_{z_2z_1}$, notice that by the definition of the path P_z we have the following non-edges: xq_1, xq_{k-1}, z_1q_2, z_2q_k and q_iq_{i+2}, for $1 \leq i \leq k - 2$. If $z_1q_k \notin E(H_I)$ then we have the following odd cycle:
    $$C_{k+2} = (x, z_1)(z_1, q_1)(q_1, q_2) \dots (q_{k-1}, q_k)(q_k, x)(x, z_1).$$
    In case $z_1q_k \in E(H_I)$ we have the following three cases to consider: If $z_1q_i \in E(H_I)$, $1 \leq i \leq k$ then we have the following even path ($r = k + 3$):
    $$P_{z_2z_1} = (z_2, z_1)(z_1, q_k)(q_k, z_1)(z_1, q_{k-2}) \dots (q_3, z_1)(z_1, q_1)(q_1, z_1)(z_1, x).$$

If $z_1q_i \notin E(H_I)$, $z_1q_{i+1}, z_1q_{i+2}, \dots z_1q_k \in E(H_I)$, and i is an even number, $1 < i < k$, then we have the following even path ($r = k+3$):

$$P_{z_2z_1} = (z_2, z_1)\underbrace{(z_1, q_k)(q_k, z_1)(z_1, q_{k-2})\dots(z_1, q_{i+1})(q_{i+1}, q_i)}_{k-i+1}P_{i+1},$$

where $P_{i+1} = (q_i, q_{i-1})(q_{i-1}, q_{i-2})\dots(q_1, z_1)(z_1, x)$.
If $z_1q_i \notin E(H_I)$, $z_1q_{i+1}, z_1q_{i+2}, \dots z_1q_k \in E(H_I)$ and i is an odd number, $1 < i < k$, then we have the following odd cycle in B'_I:

$$C_{k+2} = (x, z_1)\underbrace{(z_1, q_{k-1})(q_{k-1}, z_1)(z_1, q_{k-3})\dots(z_1, q_{i+1})(q_{i+1}, q_i)}_{k-i}P_{i+2},$$

where $P_{i+2} = (q_i, q_{i-1})(q_{i-1}, q_{i-2})\dots(q_1, z_1)(z_1, x)(x, z_1)$.

– *Case 2.3: $wz_1 \in E(H_I)$ and $wz_2 \in E(H_I)$.*
In this case we prove that if there is an even path P_z which connects (x, z_1) and (x, z_2) then there is either (i) a path P_{xz_1} (resp. P_{xz_2}) on r_1 vertices where r_1 is an even number which connects (x, w) and (w, z_1) (resp. (w, z_2)) or (ii) a path P_{wz} on r_2 vertices where r_2 is an even number which connects (w, z_1) and (w, z_2). In both cases the result follows since if (i) holds then the following odd cycle appears in B'_I (notice that $z_1v, z_2v \notin E(H_I)$):

$$C_{\ell+r_1+5} = P^T_{xv}P_{xz_1}(z_1, w)(w, v)(v, w),$$

and if (ii) holds then we have the following odd cycle:

$$C_{r_2+3} = (v, w)P_{wz}(z_2, w)(w, v)(v, w).$$

To justify the existence of the paths P_{xz_1} and P_{xz_2}, observe first that if $wq_1 \notin E(H_I)$ and $wq_k \notin E(H_I)$ then we have the following even path ($r_1 = k+5$):

$$P_{xz_1} = (x, w)\underbrace{(w, x)(x, q_k)(q_k, q_{k-1})\dots(q_2, q_1)(q_1, z_1)(z_1, w)}_{k+3}(w, z_1). \tag{1}$$

If $wq_i \in E(H_I)$, $1 \le i \le k$, then we have the following even path ($r = k+1$):

$$P_{xz_1} = (x, w)\underbrace{(w, q_{k-1})(q_{k-1}, w)(w, q_{k-3})(q_{k-3}, w)\dots(q_4, w)(w, q_2)(q_2, w)}_{k-1}(w, z_1).$$

Now in all other cases let $q_jw \notin E(H_I)$ and $q_1w, q_2w, \dots, q_{j-1}w \in E(H_I)$, and let $q_iw \notin E(H_I)$ and $q_{i+1}w, q_{i+2}w, \dots, q_kw \in E(H_I)$, $1 \le j \le i \le k$. Depending on the values of i and j, we have the following four cases to consider:
• If i is an odd number and j is an even number then we have the following odd cycle in B'_I:

$$C_{k+2} = \begin{cases} (x, w)P_j\underbrace{(q_j, q_{j+1})\dots(q_{i-1}, q_i)}_{i-j}P_{k-i}(w, x)(x, w), & \text{if } i < k \\ (x, w)P_j\underbrace{(q_j, q_{j+1})\dots(q_{k-1}, q_k)}_{k-j}(q_k, x)(x, w), & \text{if } i = k \end{cases}$$

where $P_j = (w,q_1)(q_1,w)(w,q_3)(q_3,w)\dots(q_{j-3},w)(w,q_{j-1})(q_{j-1},q_j)$
and $P_{k-i} = (q_i,q_{i+1})(q_{i+1},w)(w,q_{i+3})\dots(q_{k-3},w)(w,q_{k-1})(q_{k-1},w)$.

• If i is an odd number and j is an odd number then we have the following even path:

$$P_{xz_1} = \begin{cases} (x,w)P_{k-i}\underbrace{(q_i,q_{i-1})\dots(q_{j+1},q_j)}_{i-j}P_{j-1}(w,z_1), & \text{if } 1<j\le i<k \\ (x,w)(w,x)(x,q_k)\underbrace{(q_k,q_{k-1})\dots(q_{j+1},q_j)}_{k-j}P_{j-1}(w,z_1), & \text{if } 1<j \text{ and } i=k \\ (x,w)P_{k-i}\underbrace{(q_i,q_{i-1})\dots(q_2,q_1)}_{i-1}(q_1,z_1)(z_1,w)(w,z_1), & \text{if } j=1 \text{ and } i<k \end{cases}$$

where $P_{j-1} = (q_j,q_{j-1})(q_{j-1},w)(w,q_{j-3})\dots(w,q_4)(q_4,w)(w,q_2)(q_2,w)$
and $P_{k-i} = (w,q_{k-1})(q_{k-1},w)(w,q_{k-3})\dots(q_{i+3},w)(w,q_{i+1})(q_{i+1},q_i)$.
Note that the first path has $k+1$ vertices, whereas the next two paths have $k+3$ vertices, respectively. Recall that if $j=1$ and $i=k$, then the corresponding path is described in Equation 1.

• If i is an even number and j is an even number then we have the following even path ($r_1 = k+3$):

$$P_{xz_2} = (x,w)P_j\underbrace{(q_j,q_{j+1})\dots(q_{i-1},q_i)}_{i-j}P_{k-i+1}(w,z_2),$$

where $P_j = (w,q_1)(q_1,w)(w,q_3)(q_3,w)\dots(q_{j-3},w)(w,q_{j-1})(q_{j-1},q_j)$
and $P_{k-i+1} = (q_i,q_{i+1})(q_{i+1},w)(w,q_{i+3})\dots(q_{k-2},w)(w,q_k)(q_k,w)$.

• If i is an even number and j is an odd number then we have the following even path ($r_2 = k+3$):

$$P_{wz} = \begin{cases} (w,z_1)(z_1,w)P_{j-1}\underbrace{(q_j,q_{j+1})\dots(q_{i-1},q_i)}_{i-j}P_{k-i+1}(w,z_2), & \text{if } 1<j \\ (w,z_1)(z_1,q_1)\underbrace{(q_1,q_2)\dots(q_{i-1},q_i)}_{i-1}P_{k-i+1}(w,z_2), & \text{if } j=1 \end{cases}$$

where $P_{j-1} = (w,q_2)(q_2,w)(w,q_4)(q_4,w)\dots(q_{j-3},w)(w,q_{j-1})(q_{j-1},q_j)$
and $P_{k-i+1} = (q_i,q_{i+1})(q_{i+1},w)(w,q_{i+3})\dots(q_{k-2},w)(w,q_k)(q_k,w)$.

□

Now we show that adding the incompatibilities at the second forall-loop starting at line 12 does not create an odd cycle, if we skip the first for-all loop starting at line 10.

Lemma 15. *Assume that B_I is bipartite. If xv is a conflict at step I, then there is no path on even number of vertices connecting (v,u_1) and (v,u_2) in B'_I, for every pair of vertices u_1,u_2 such that $u_1,u_2 \in N_{H_I}(v)$, $u_1,u_2 \notin N_{H_I}[x]$ and xu_1, xu_2 are marked non-edges.*

Proof. Notice that the incompatibilities $(x,w)(w,u_1)$ and $(x,w)(w,u_2)$ are present in B'_I since xu_1 and xu_2 are marked non-edges. Thus if we swap vertices x and v, and if we set u_1 and u_2 to be z_1 and z_2, respectively, then the proof is similar (identical) to that of Lemma 14. □

Thus we have seen that each of the two forall-loops maintains the bipartite graph if we skip the other for-all loop. Let us now show that together they do not create a problem.

Lemma 16. *Assume that B_I is bipartite. If xv is a conflict at step I, then there is no path on even number of vertices connecting (x, z_1) and (u_1, v) in B'_I, for every pair of vertices z_1, u_1 such that $u_1 \in N_{H_I}(v)$, $u_1 \notin N_{H_I}[x]$, xu_1 is a marked non-edge and $z_1 \in N_{H_I}(x)$, $z_1 \notin N_{H_I}[v]$.*

Proof. Assume for the sake of contradiction that there is an even path P_{zu} connecting them. By Lemma 11 this path has the following form:

$$P_{zu} = (x, z_1)(z_1, y_1) \dots (y_s, u_1)(u_1, v),$$

where $s \geq 1$ is an odd number. Hence the path P_{zu} contains $s + 3$ vertices. We will prove that in this case there is an odd cycle in B'_I which is a contradiction since B'_I is a bipartite graph. First we prove that if $z_1u_1 \in E(H_I)$ then we have the following odd cycle in B'_I by the fact that $z_1 \notin N_{H_I}[v]$ and $u_1 \notin N_{H_I}[x]$:

$$C_{s+6} = P_{zu}(v, u_1)(u_1, z_1)(z_1, x)(x, z_1).$$

Notice that if $z_1u_1 \in E(H_I)$ and $s = 1$ then there no path on even number of vertices connecting (x, z_1) and (u_1, v) in B'_I. Thus we continue by knowing that $z_1u_1 \notin E(H_I)$ and $s \geq 1$. Notice also that by Observation 12 there is a vertex w which induces a path P_{xv} with $\ell + 3$ vertices in B'_I, where ℓ is an even number. We distinguish four cases according to whether w is adjacent or not to z_1 or/and v_1:

- *Case A: $wz_1 \notin E(H_I)$ and $wu_1 \notin E(H_I)$.*
  It is easy to see that the following odd cycle appears in B'_I:

  $$C_{\ell+s+6} = P_{zu}P^T_{xv}(x, z_1).$$

- *Case B: $wz_1 \in E(H_I)$ and $wu_1 \notin E(H_I)$.*
  Here we have two cases to consider according to whether or not $wy_1 \in E(H_I)$. In both cases we prove that an odd cycle appears in B'_I. First notice that if $wy_1 \notin E(H_I)$ then the following odd cycle occurs in B'_I:

  $$C_{s+4} = (w, z_1)\underbrace{(z_1, y_1)(y_1, y_2) \dots (y_{s-1}, y_s)(y_s, u_1)}_{s+1}(u_1, v)(v, w)(w, z_1).$$

  Also notice that if $wy_i \in E(H_I)$ for $1 \leq i \leq s$ then we have the following odd cycle:

  $$C_{s+\ell+6} = (x, w)\underbrace{(w, y_1)(y_1, w)(w, y_3) \dots (w, y_s)}_{s}(y_s, u_1)(u_1, v)P^T_{xv}(x, w).$$

  In case $wy_1, wy_2, \dots wy_{i-1} \in E(H_I)$ and $wy_i \notin E(H_I)$ then we distinguish two cases according to the value of i. If i is an odd number then the following odd cycle appears in B'_I:

  $$C_{s+4} = (w, z_1)(z_1, w)P_{i-1}\underbrace{(y_i, y_{i+1}) \dots (y_s, u_1)}_{s-i+1}(u_1, v)(v, w)(w, z_1),$$

where $P_{i-1} = (w, y_2)(y_2, w)(w, y_4) \dots (y_{i-3}, w)(w, y_{i-1})(y_{i-1}, y_i)$.

Otherwise (i is an even number) we have:

$$C_{s+\ell+6} = (x, w)P_i \underbrace{(y_i, y_{i+1}) \dots (y_s, u_1)}_{s-i+1}(u_1, v)P^T_{xv}(x, w),$$

where $P_i = (w, y_1)(y_1, w)(w, y_3) \dots (y_{i-3}, w)(w, y_{i-1})(y_{i-1}, y_i)$.

- *Case C: $wz_1 \in E(H_I)$ and $wu_1 \notin E(H_I)$.*
  This case is similar (symmetric) to the previous one. By swapping vertices z_1 and u_1 we conclude to the same result; notice that the incompatibility $(x, w)(w, u_1)$ is present in B'_I since xu_1 is a marked non-edge of H_I.

- *Case D: $wz_1 \in E(H_I)$ and $wu_1 \in E(H_I)$.*
  Here we obtain the following odd cycle in B'_I:

$$C_{\ell+9} = (z_1, w)(w, v)P^T_{xv}(x, w)(w, u_1)(u_1, w)(w, z_1)(z_1, w).$$

□

Now we are ready to reach the desired result.

Lemma 17. *At each step of the algorithm B_I is a bipartite graph.*

Proof. At the beginning of the algorithm, we know that $B_G \cup B_x$ is bipartite, and that all possible conflicts of G_x are contained in L. Assume that B_I is a bipartite graph. We show that B_{I+1} is also a bipartite graph. At step I, we have two cases to consider. If $H_{I+1} = H_I$, then this is because $B_{I+1} = B_I \cup C_{H_I}(xv)$ is a bipartite graph. Let H_{I+1} be obtained from H_I by adding fill edge xv. Then B_{I+1} is obtained from B_I by adding an isolated edge $(x, v)(v, x)$, and some incompatibilities incident to the endpoints of this edge, implied by non-edges outside of L. These incompatibilities are added by the first for–loop at line 10 and the second for–loop at line 12.

For the first for–loop if the set $N_{H_{I+1}}(x) \cap N_{H_{I+1}}[v]$ contains only one vertex, say z_1, then there is an even cycle in B_{I+1} formed by the vertices $(x, v), (v, z_1), (v, x), (z_1, v)$ and no odd cycle is created in B_{I+1}. It is easy to see that the same argument (for one vertex u_1) holds for the second for-loop. Now in general notice that any odd cycle created in B_{I+1} will still be an odd cycle in B'_{I+1} by Observation 12 since $B_{I+1} \subseteq B'_{I+1}$. But since B'_I is bipartite any odd cycle in B'_{I+1} can be created only if the conditions of Observation 10 are true. Thus by Lemmata 14–16 we justify that the corresponding cases cannot exist and therefore B'_I remains bipartite by applying the two for–loops, i.e., B'_{I+1} is a bipartite graph. Hence the result follows from the fact that $B_{I+1} \subseteq B'_{I+1}$. □

Theorem 18. *The graph H returned by Algorithm MCC is a minimal comparability completion of G_x.*

Proof. First we show that H is a comparability completion of G_x. During the algorithm, every time a new incompatible pair is created, the corresponding incompatibility is added to B_I unless it is implied by a non-edge of L. Incompatibilities implied by members of L that remain non-edges are added one by one until L is empty. At the end of the algorithm, the graph B

contains all incompatibilities implied by the non-edges in H, since $L = \emptyset$. Thus B is the correct incompatibility graph of H, i.e., $B = B_H$. Since B_H is bipartite graph by Lemma 17, the resulting graph H is a comparability graph by Theorem 1.

Now we want to prove that H is minimal, that is, if any subset of the fill edges is removed the remaining graph is not comparability. Recall that at any step of the algorithm we do not remove any edges from the graph B_I (see also Lemma 7). Assume for the sake of contradiction that there is a subset F of the fill edges such that $H' = H - F$ is a comparability graph. First note that $B_{H'}$ is obtained from B_H by removing the vertices (x, u) and (u, x), and then adding the set $C_{H'}(xu)$, for every $xu \in F$. Let I be the earliest step in which Algorithm MCC adds a fill edge $xv \in F$. Thus no non-edge of G_x belonging to F has been processed before step I, and H_I is a subgraph of H'. Furthermore, B_I does not contain any edge belonging to $\bigcup_{xu \in F} C_{H'}(xu)$, and B_I does not contain any pair of vertices (x, u) and (u, x), for $xu \in F$. Thus B_I is a subgraph of $B_{H'}$. Now, observe that for each $xu \in F$, $C_{H_I}(xu) \subseteq C_{H'}(xu)$, since $N_{H_I}(x) \subseteq N_{H'}(x)$. In particular, $C_{H_I}(xv) \subseteq C_{H'}(xv)$. Since xv is a non-edge of H', all edges of $C_{H'}(xv)$ are present in $B_{H'}$. Therefore $B_I \cup C_{H_I}(xv)$ is a subgraph of $B_{H'}$. In Algorithm MCC, at step I, we know that $B_I \cup C_{H_I}(xv)$ contains an odd cycle, otherwise xv would not be a fill edge. Since it is not possible to remove an odd cycle by adding edges or vertices, this means that there is an odd cycle in $B_{H'}$. This gives the desired contradiction, because by Theorem 1 H' cannot be a comparability graph as assumed. □

5 Time required to compute minimal comparability completions

Let G be an arbitrary graph on n vertices and m edges. First we prove the following observation.

Observation 19. *The incompatibility graph B_G of a given graph G has $O(mn)$ edges.*

Proof. Let G be a graph on n vertices and m edges, and let B_G be its incompatibility graph. By definition B_G has precisely $2m$ vertices. Clearly B_G contains m edges of the form $(a, b)(b, a)$. For the other edges of B_G (incompatibilities) it is easy to see that each edge of G (two vertices of B_G) can define at most $O(n)$ incompatibilities in B_G since they are induced by the neighbors of its endpoints in G. Thus B_G has $O(nm)$ edges. □

Now we are ready to give the time bounds of the Algorithm MCC.

Lemma 20. *Given a comparability graph G and its incompatibility graph B_G, Algorithm MCC computes a minimal comparability completion of G_x in $O(n^2m)$ time.*

Proof. Let G be a comparability graph on n vertices and m edges, and let B_G be its incompatibility graph. Since only non-edges incident to x are processed, $|L| = O(n)$, and since non-edges removed from L are never reinserted in L, the algorithm has $O(n)$ steps. By Observation 19 B_G has $O(nm)$ edges. Since $|N_x| = O(n)$, B_x has $O(n)$ vertices and thus $O(n^2)$ edges. At each of the $O(n)$ steps, we can add at most $O(n)$ edges to B since $|C_H(xv)| = O(n)$ for each $xv \in L$. Thus at all steps B has $O(nm)$ edges. What dominates our time complexity is to check whether or not $B \cup C_H(xv)$ is bipartite. This check can be done in time linear in the size of B, namely $O(nm)$. Therefore, each step of the algorithm requires $O(nm)$ time, which gives a total running time of $O(n^2m)$. □

We point out that given an incompatible pair $((a,b)(b,c))$ of G there is an $O(n+m)$ time algorithm deciding whether its incompatibility graph has an odd cycle [7]. However, it is not straightforward to use this result for checking whether the graph $B_I \cup C_{H_I}(xv)$ of Algorithm MCC is bipartite in $O(n+m)$ time, since at each step of the algorithm, B_I is merely a subgraph of B_{H_I}, and B_I is not necessarily equal to B_{H_I} before the last step. The following result follows from Lemma 3, Lemma 20, and Algorithm MCC.

Theorem 21. *There is an algorithm for computing a minimal comparability completion of an arbitrary graph G in $O(n^3m)$ time.*

6 Concluding Remarks

In this paper, we have shown that minimal comparability completions of arbitrary graphs can be computed in polynomial time. Comparability graphs can be recognized in time $O(n^{2.38})$ [14], and even the straight forward $O(n^3m)$ running time analysis of our algorithm for computing minimal comparability completions is thus comparable to the time complexity of recognizing comparability graphs. As a comparison, both chordal and interval graphs can be recognized in linear time; the best known time for minimal chordal completions is $O(n^{2.38})$ [6], and for minimal interval completions is $O(n^5)$ [5].

Although minimal comparability completions can be computed in polynomial time, this does not imply that the following problem is solvable in polynomial time: Given a comparability completion H of an arbitrary graph G, is H a minimal comparability completion of G? We would like to know whether this problem can be solved in polynomial time. In fact, it would be very useful and interesting to obtain a characterization of minimal comparability completions.

There are minimal comparability completions which the algorithm given in this paper cannot compute. For the goal of using minimal comparability completions in the search for minimum comparability completions, we would need an algorithm that is able to generate any possible minimal comparability completion of a given graph. We leave it an open problem to design an algorithm that is both efficient and able to do this.

References

[1] A. Berry, P. Heggernes, and Y. Villanger. A vertex incremental approach maintaining chordality. *Discrete Math.*, 306(3):318–336, 2006.

[2] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, 1980.

[3] S. L. Hakimi, E. F. Schmeichel, and N. E. Young. Orienting graphs to optimize reachability. *Information Processing Letters*, 63(5):229–235, 1997.

[4] P. Heggernes and F. Mancini. Minimal split completions of graphs. In *LATIN 2006: Theoretical Informatics*, pages 592–604. Springer Verlag, 2006. LNCS 3887.

[5] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Minimal interval completions. In *Algorithms - ESA 2005*, pages 403 – 414. Springer Verlag, 2005. LNCS 3669.

[6] P. Heggernes, J. A. Telle, and Y. Villanger. Computing minimal triangulations in time $O(n^\alpha \log n) = o(n^{2.376})$. In *Proceedings of SODA 2005 - 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 907–916, 2005.

[7] D. Kratsch, R. M. McConnell, K. Mehlhorn, and J. P. Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. *SIAM J. Comput.*, 2006. To appear.

[8] D. Kratsch and J. P. Spinrad. Between $O(nm)$ and $O(n^\alpha)$. In *Proceedings of SODA 2003 - 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 709–716, 2003.

[9] D. Kratsch and J. P. Spinrad. Minimal fill in $o(n^3)$ time. *Discrete Math.*, 306(3):366–371, 2006.

[10] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Disc. Appl. Math.*, 113:109–128, 2001.

[11] I. Rapaport, K. Suchan, and I. Todinca. Minimal proper interval completions. In *Proceedings of WG 2006 - 32nd International Workshop on Graph-Theoretic Concepts in Computer Science*, 2006. To appear.

[12] F. S. Roberts. *Graph Theory and Its Application to Problems of Society.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1978.

[13] D. Rose, R.E. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266 – 283, 1976.

[14] J. Spinrad. On comparability and permutation graphs. *SIAM J. Comput.*, 14:658 – 670, 1985.

[15] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.