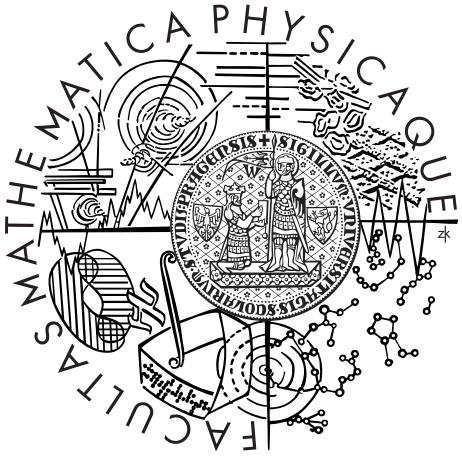
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Anna Chejnovská

Optimisation using graph searching on special graph classes

Department of Applied Mathematics

Supervisor of the bachelor thesis: Mgr. Tomáš Gavenčiak

Study programme: Mathematics

Specialization: General Mathematics

Prague 2015

I would like to thank my supervisor for his feedback on my work and patience. I would also like to thank Petr Marhoun, Jan Brandejs and Jan Pipek for proof reading. Finally I would like to thank Anna Polášková and Zuzana Vlčková for helping me to choose the right graph design and for moral support.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... on ..... Author signature

Název práce: Optimalizace grafovým prohledáváním na speciálních třídách grafů

Autor: Anna Chejnovská

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Tomáš Gavenčiak, Katedra aplikované matematiky

Abstrakt: Pro graf definujeme minimální pokrytí cestami jako nejmenší množinu vrcholově disjunktních cest, které pokrývají všechny vrcholy grafu. Tento problém je jedním z běžných zobecnění známého problému hledání Hamiltonovské cesty v grafu. V této práci vyjdeme ze článku Corneil et al. (2013), kde byl představen certifikující algoritmus pro hledání minimálního pokrytí cestami na cocomparability grafech (doplněk grafu relace neostrého částečného uspořádání). Tento algoritmus nejprve představíme, poté experimentálně prozkoumáme jeho robustnost vůči pěti operacím na hranách a vrcholech. Dopad těchto operací na velikost minimálního pokrytí cest rozebereme také teoreticky.

Klíčová slova: cocomparability grafy, minimální pokrytí cestami, prohledávající algoritmy, robustnost

Title: Optimisation using graph searching on special graph classes

Author: Anna Chejnovská

Department: Department of Applied Mathematics

Supervisor of the bachelor thesis: Mgr. Tomáš Gavenčiak,, Department of Applied Mathematics

Abstract: For a given graph we define the minimum path cover as a minimum cardinality set of vertex disjoint paths covering all the vertices of the graph. This problem is one of the usual generalization of the Hamiltonian path problem. In this thesis we based our work on a paper Corneil et al. (2013) presenting a certifying algorithm for the minimal path cover problem on cocomparability graphs (the complement of graph of strict partial order). We first introduce this algorithm an then we experimentally examine its robustness to five operations on edges and vertices of the graph. We also analyse the impact of these operation on the size of the minimal path cover theoretically.

Keywords: cocomparability graphs, minimal path cover, searching algorithms, robustness

Contents

Chapter 1

Introduction

The Hamiltonian path problem is one of the most well know NP-complete problems. It is about finding a path which visits each vertex exactly once. A cycle containing all the vertices from a graph is called a Hamiltonian cycle of the graph. It is named after a game based on the constructing a cycle on all of the vertices of dodecahedron invented by Sir William Rowan Hamilton in 1857. In fact a puzzle with the same motive, i.e. finding a Hamiltonian path was studied by Euler in 1759 as an knight's tour on the chessboard. These puzzles and games are more discussed in Bollobas (1998).

There are few different optimization versions of this problem such as the longest path problem (discussed in Mertzios and Corneil (2012)) or the minimal path cover. These problems are still too hard on general graphs so we look at special graph classes, namely cocomparability graphs.

The minimum path cover problem is to find the minimum number of vertex disjoint paths $\mathcal{P}$ such that each vertex of the graph belongs to exactly one path from $\mathcal{P}$. This problem has a very elegant algorithm for interval graphs which was presented in Arikati and Pandu Rangan (1990) and Damaschke (1993). This algorithm was adjusted in Corneil et al. (2013) that it can solve this problem even on cocomparability graphs. The algorithm provides a certificate which tells us that the result wasn't compromised by a bug. Another advantage of the certifying algorithm is that it also checks if the input graph was a cocomparability graph.

We implement this algorithm and examine what happens if we slightly spoil the cocomparability by adding/removing few edges and vertices and contracting few edges. In other words we examine robustness of the algorithm which is ability to cope with abnormalities in input.

This work is organized as follows. In Chapter 2, we provide the necessary preliminaries, notation and basic properties including the basic search algorithms. In Chapter 3, we summarize known results on this topic. In Chapter 4 most of our theoretical results about modifying the cocomparability graphs take place following by experimental results in Chapter 5. Chapter 6 belongs to the description to attachments on the cd. The last Chapter 7 contains a conclusion of this work.

Chapter 2

Preliminaries

2.1 Definitions and basic properties

The basic definitions we will take mainly from Bollobas (1998), Corneil et al. (2013) and Corneil et al. (2008).

A *graph* G is an ordered pair of disjoint sets (V, E) such that E is a subset of the set of unordered pairs of V. The set V is the set of *vertices* and E is the set of edges. An edge between vertices u and v is denoted by (u, v). The set of *neighbouring vertices* of v is denoted by $N(v)$ and it contains all of the vertices u from V such that $(u, v) \in E$. We consider only finite graphs, that is, V and E are always finite. All our graphs are *simple graphs* as well, which means they contain no *loops*, i. e. an edge joining a vertex to itself, neither do they contain *multiple edges*, that is several edges joining the same two vertices.

If we have a *strict partial order*, that is a binary relation $<$ over a set S which is irreflexive ($a \not< a$), transitive (if $a < b$ and $b < c$ then $a < c$) and asymmetric (if $a < b$ then $b \not< a$), we can define a *comparability graph* $G = (S, E)$, where $E = \{(u, v) \in \binom{S}{2} : u < v \text{ or } v < u\}$.

In this work we observe especially the class of *cocomparability graphs* which are complements of comparability graphs. For a graph $G = (V, E)$ the *complement* of G is $\bar{G} = (V, \binom{V}{2} \backslash E)$.

Definition 1 (cocomparability ordering)**.** *Let $G = (V, E)$ is a graph, then cocomparability ordering is an ordering σ of V, such that for all triples of vertices $x <_\sigma y <_\sigma z$, if $(x, z) \in E$ then $(x, y) \in E$ or $(y, z) \in E$. Other names for cocomparability ordering are umbrella-free ordering and cocomp ordering.*

Following lemma was first observed in Kratsch and Stewart (1993).

Lemma 1 (cocomparability graph characterization)**.** *$G = (V, E)$ is a cocomparability graph if and only if there is an umbrella-free ordering σ of V.*

Proof. Let's have a comparability graph and three of its vertices $x < y < z$. From transitivity we have that if there isn't (x, z) at least one of the edges (x, y) and (y, z) couldn't be in the graph as well. If we project this situation to the complement graph we get the statement from the lemma.

Let's have a graph G which is not cocomparability and umbrella-free ordering σ. Thus the complement $\bar{G}$ is not comparability. That means that situation on

triple of vertices $x <_\sigma y <_\sigma z$ with only edges (x, y) and (y, z) can occur so let's have this situation. For the complement graph that means that there is an edge (x, z) but neither (x, y) nor (y, z). Therefore σ can't be cocomp order.

❑

Throughout this work we will apply five different operations on the graphs. Let's have a graph $G = (V, E)$ with a vertex ordering σ. By *addition of edge e* where $e \notin E$ we obtain graph

$$G_{e+} = (V, E \bigcup \{e\}).$$

The opposite operation is *removal of edge e* where $e \in E$. By this process we obtain a graph

$$G_{e-} = (V, E \backslash \{e\}).$$

Another one is *addition of a universal vertex v* which changes G into

$$G_{v+} = (V \bigcup \{v\}, E \bigcup_{w \in V} (w, v)).$$

Following operation is *removal of vertex v* which creates from G graph

$$G_{v-} = (V \backslash \{v\}, E \backslash \bigcup_{w \in N(v)} (w, v)).$$

The last one is *contraction of an edge e* where $e = (x, y), x <_\sigma y$ which concludes in a graph

$$G_c = (V/\{y\}, E \bigcup_{w \in N(y), (w,x) \notin E} (w, x) \backslash \bigcup_{z \in N(y)} (z, y)).$$

Let $pc(G)$ denote the path cover number of graph G, the smallest size of any path cover. Let S be a proper subset of V and $c(G \backslash S)$ number of connected components of $G \backslash S$. Then we defined scattering number as

$$sc(G) = \max\{c(G \backslash S) - |S| : S \subset V, c(G \backslash S) > 1\}.$$

Lemma 2. *For an arbitrary graph $G = (V, E)$ holds an inequality $pc(G) \geq sc(G)$.*

Proof. We have to cover by paths each of the connected components so $pc(G \backslash S) \geq c(G \backslash S)$. Every vertex of S can connect at most two paths from the minimal path cover of $G \backslash S$ thus $pc(G) \geq pc(G \backslash S) - |S| \geq c(G \backslash S) - |S|$. This inequality holds for every S so $pc(G) \geq sc(G)$.

❑

From Deogun et al. (1997) we know that for cocomparability graphs holds the equality $pc(G) = sc(G)$ which is one of the principal facts we use below.

2.2 Algorithms

Graph searching is the problem of visiting all the vertices of the graph. In this work, we use several graph searching algorithms, so we will introduce them first. All of them are discussed in Corneil and Krueger (2008) in greater detail.

2.2.1 Lexicographic depth-first with + sweep

This algorithm is based on the Depth-first search (DFS) and we denote it LDFS. In DFS if the algorithm is in the vertex v and you want to visit its neighbours, you choose their order arbitrarily. The main change of LDFS against the DFS is that we have an exact rule how to pick the next neighbour – according to the lexicographic labels.

We use a slightly different version in this work LDFS+. At the input we get an ordering π of the vertices and everytime we need a starting node we chose the rightmost unvisited vertex from π. So our starting vertex from Algorithm 1 is the last vertex from π.

Algorithm 1: Lexicographic depth-first search +

Input: Graph $G = (V, E)$ of the size n with an ordering π
Output: Order σ where σ_i is ith vertex visited
assign the label ϵ to all vertices
for $i \leftarrow 1$ **to** n **do**
    pick an unvisited vertex v in π among unvisited vertices with the lexicographically largest label
    $\sigma_i \leftarrow v$
    foreach $w \in N(v)$, *w is unvisited* **do**
        prepend i to *label*(w)
    end
end
return *Order of vertices* σ

2.2.2 Rightmost neighbour search

The rightmost neighbour search (RMN) works as the name suggests. Input is graph G and ordering σ. The algorithm takes the rightmost vertex of a node ordering σ and continues to its rightmost neighbour according to this ordering.

Algorithm 2: Rightmost neighbour search

Input: Graph $G = (V, E)$ of the size n with vertex ordering σ
Output: Array τ which contains a path cover consisting of p paths, each stored consecutively in τ; τ_i is the ith vertex visited

```
τ_1 ← v ← σ_n
p ← 1
for i ← 2 to n do
    if there is unvisited neighbour of v then
        w ← the rightmost such vertex as ordered by σ
    else
        if i < n then
            w ← the rightmost unvisited vertex as ordered by σ
            p ← p + 1
        end
    end
    τ_i ← v ← w
end
return Path cover stored in τ
```

Chapter 3

Known results

We base our work on the LDFS+-based certifying algorithm for the minimum path cover problem on cocomparability graphs from Corneil et al. (2013). We present their certifying algorithm first and then the whole minimal path cover certifying algorithm.

As we mentioned in Chapter 2 for cocomparability graphs we have the equality: $pc(G) = sc(G)$. So to show that given path covering $\mathcal{P}$ has the minimum cardinality, it suffices to find a proper subset of vertices S such that $|\mathcal{P}| = c(G \backslash S) - |S|$.

Algorithm 3: CERTIFY

Input: An ordered path cover $\mathcal{P}$ stored in τ
Output: Vertex separator S, and the connected components of $G \backslash S$
$S \leftarrow \emptyset$
$\mathcal{Q} \leftarrow \mathcal{P}$
$p \leftarrow 1$
while *there exists a connected pair of paths* $(P_i, P_j), i < j$ *in* $\mathcal{Q}$ **do**
  Choose edge (x, y) where $x \in P_i, y \in P_j, P_i, P_j \in Q$ such that vertex x is as rightmost as possible according to τ
  $S \leftarrow S \bigcup \{x\}$
  In Q we replace P_i by the subpaths of P_i caused by the removal of x
  $p \leftarrow p + 1$
end
return $S, \mathcal{Q}$

Computing a scattering number is generally NP-complete (for example for bipartite graphs) but for cocomparability graphs there exists a polynomial algorithm as it is discussed in Kratsch et al. (1994).

The proof of correctness of the certifying algorithm for the minimal path cover on cocomparability graphs can be found at Corneil et al. (2013). We quote here the two fundamental theorems from this paper:

Theorem 3. *If σ is an LDFS cocomp order, and τ is RMN(σ), then S and $\mathcal{Q}$ produced by Algorithm CERTIFY satisfy*

- $|\mathcal{P}| = |\mathcal{Q}| - |S|$ *(where $\mathcal{P}$ is a set of paths in τ),*

- *the connected components of* $G = (V \backslash S, E \backslash \bigcup_{s \in S, v \in V}(s, v))$ *are stored as paths in* $\mathcal{Q}$.

Theorem 4. *Algorithm MPC applied on the input* (G, π) *computes the minimal path cover or answers that* π *is not an umbrella-free ordering. The algorithm can be implemented to run in time* $O(\min\{n^2, n + m \log\log n\})$*, where* $n = |V|$ *and* $m = |E|$ *of cocomparability graph* $G = (V, E)$.

Algorithm 4: MPC

Input: Cocomparability graph $G = (V, E)$ and umbrella-free order π
Output: A minimal path cover $\mathcal{P}$ of G and certifying separator S or an error message that π is not a umbrella-free order

$\sigma \leftarrow$LDFS+(π)
$\mathcal{P} \leftarrow$RMN(σ)
$(\mathcal{S}, \mathcal{Q}) \leftarrow$CERTIFY$(\mathcal{P})$
if $|\mathcal{P}| = |\mathcal{Q}| - |\mathcal{S}|$ **then**
  return $(\mathcal{P}, \mathcal{S})$
else
  return *'π is not an umbrella-free order'*
end

Chapter 4

Theoretical results

4.1 Description of the deformations

We apply five different operations on our graphs: adding edges, removing edges, adding universal vertices, removing vertices and edge contraction.

We know that the algorithm is optimal on cocomparability graphs and we want to find out which of these modifications violate cocomparability. The certificate verifies if $|\mathcal{P}| = |\mathcal{Q}| - |S|$. If the cocomparability is spoiled, the equality doesn't have to hold so we define a *gap* as a difference between the sides of this equation to observe how much the algorithm is violated. Thus $gap = |\mathcal{P}| - |\mathcal{Q}| + |S|$ and for the operations which preserve cocomparability we should have all gaps zero.

We use the umbrella-free ordering property of the cocomparability graphs from Lemma 1 which holds for the each triple of vertices $x, y, z \in V, x < y < z$.

4.1.1 Removing an edge

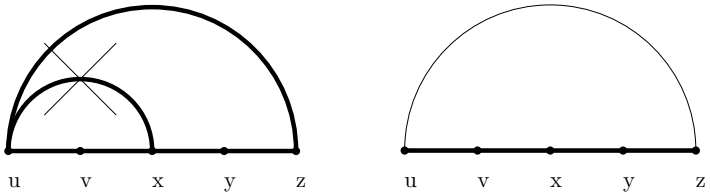
This operation can spoil cocomparability. For example circle on five vertices (C_5) with an added edge, as you can see in Figure 4.1a is cocomparability graph but C_5 is not. This case works for every permutation of the vertices. If we have a graph and order, it is sufficient if it spoils cocomparability just for the particular order.

4.1.2 Adding an edge

The edge addition can spoil cocomparability as well. Path on arbitrary number of vertices is obviously cocomparability. Like in previous case we use C_5 as an example of graph, which is not cocomparability for all permutation of vertices. We can easily obtain C_5 from path on five vertices by connecting ends of path by an edge. You can see this situation in Figure 4.1b.

4.1.3 Removing a vertex

If we remove the vertex v, the umbrella-free ordering property still holds for the triples without v and the triples with v don't exist any more. As conclusion this modification preserves cocomparability of the graph.



(a) Removing an edge (b) Adding an edge

Figure 4.1: Examples of spoiling cocomparability, where $u < v < x < y < z$

4.1.4 Adding an universal vertex

We have to place the additional vertex somewhere in the ordering first, so we put it to the end of the order. In a similar way as for the node removal, an universal vertex addition preserves cocomparability. We are interested just in the triples with the new universal vertex w. This vertex is connected by an edge to the rest of the vertices in the triple so it is umbrella-free.

4.1.5 Contracting an edge

Lemma 5. *An edge contraction preserves cocomparability.*

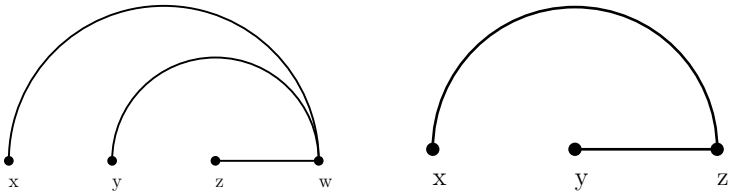
Proof. Let's have a graph with cocomp ordering σ. By contraction of an edge (u, v) we replace vertices u, v by a new one which we plant in σ to a place of the smaller of u, v according to σ.

σ would not be an umbrella-free order if there exists a triple of vertices $x < y < z$ such that (x, z) is an edge and we obtained this triple from umbrella-free triple on these vertices. Since the contraction doesn't remove edges between vertices, it had to be a triple without edges. We try to figure out how the edge (x, z) could be developed by an edge contraction. There had to be one more vertex w which disappears by the contraction. Since we want to obtain the edge (x, z), we have two possibilities of edge to contract: (z, w) or (x, w).

First we assume that the contracted edge is (z, w) (after contraction the new vertex is in the place of the smaller one according to the ordering so it has to be $z < w$). Thus we have vertices x, y, z, w with edges (z, w) and (x, w) (to gain (x, z)). But it was an umbrella-free order before the contraction, so we have to add at least one of edges (x, y), (y, w). Triple x, y, z is assumed to be without the edges so we add (y, w). With these three edges it is an umbrella-free order on those four vertices. After the contraction we get triple x, y, z with edges (x, z) and (y, z) which is an umbrella-free order as well so our attempt to destroy it was unsuccessful.

Second case is when we contract (x, w). We have to have the (z, w) or (w, z) (it depends on where the node w is placed by the ordering) as well. In accordance with the umbrella-free order, before the contraction we have to add (y, w) (or (w, y)) too. Therefore by the contraction we get triple x, y, z with edges (x, z) and (x, y) so our attempt to spoil the umbrella-free ordering failed again.

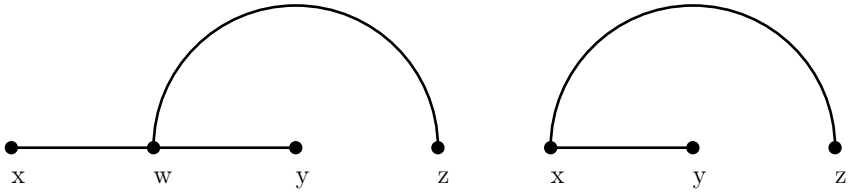
❑



(a) Situation before contraction

(b) Situation after contraction

Figure 4.2: Contraction of the edge (z, w)



(a) Situation before contraction

(b) Situation after contraction

Figure 4.3: Contraction of the edge (x, w)

4.2 Modification effect on path cover size

Here we analyse what can happen to the size of the minimal path cover, if we perform our modifications on the graph.

4.2.1 Removing an edge

The size of the minimal path cover can be increased by one ($pc(G - e) = pc(G) + 1$) if we take an edge from the path in the path cover. The situation can occur for instance as in Figure 4.4a. On the other hand it can't be increased more because the edge we are removing was a part of at most one path. Thus we have

$$pc(G) \leq pc(G - e) \leq pc(G) + 1.$$

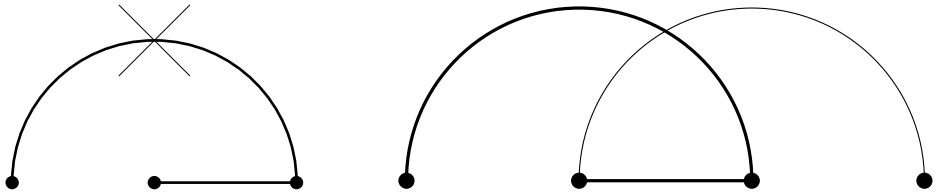
4.2.2 Adding an edge

The size of a minimal path cover can decrease by one. The added edge can connect two paths from the minimal path cover. This option can occur as we can see in Figure 4.4b. On the other hand, it can't be decreased more because we can connect at most two paths by an additional edge. Therefore we get

$$pc(G) - 1 \leq pc(G + e) \leq pc(G).$$

4.2.3 Removing a node

There are two possibilities for the removing of the node. We can take an inner vertex of a path so we can split it into two new paths, i.e. $pc(G - v) = pc(G) + 1$. However, we can take straight path P on some vertices and one spare vertex v which we connect by an edge to an inner vertex of our path P. This shape can

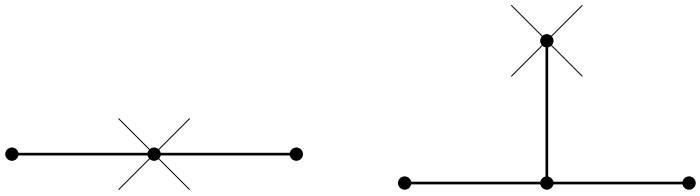


(a) $pc(G-e) = pc(G)+1$ (b) $pc(G+e) = pc(G)-1$

Figure 4.4: Examples of changing the size of the minimal path cover by adding/removing an edge

be covered by at least two paths but if we remove the v node, it is $pc(G-v) = pc(G)-1$. Both options can occur as you can see in Figure 4.5. Consequently, we obtain an inequality

$$pc(G)-1 \leq pc(G+v) \leq pc(G)+1.$$



(a) $pc(G-v) = pc(G)+1$ (b) $pc(G-v) = pc(G)-1$

Figure 4.5: Examples of changing the size of the minimal path cover by removing a node

4.2.4 Adding an universal node

The new universal vertex v is connected to the rest of the vertices. We take arbitrary ends of two different paths from the minimal path cover and connect them through v, but we can't connect more than two paths through v. Thus for a graph with a Hamiltonian path, the size of the minimum path cover remains one and if it doesn't have a Hamiltonian path the number of paths from the minimum path cover is decreased by one. In this case we actually gain an equality

$$pc(G+v) = \min\{1, pc(G)-1\}.$$

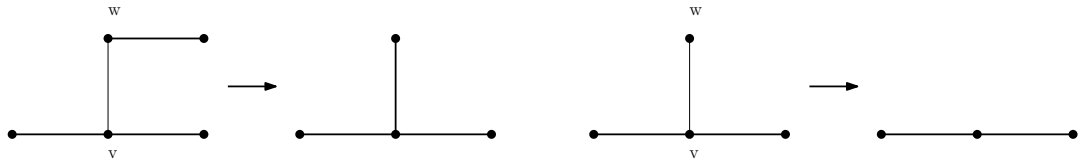
4.2.5 Edge contraction

There are several different cases of edge contraction. First we consider contraction of an edge between two nodes from two different paths. The nodes can be end points or inner vertices of the particular path.

Let's take two endpoints from different paths. This case cannot happen because there can't be an edge to contract (otherwise the two different paths would be one longer path).

Let's take one inner vertex v and one end point w. If we contract the edge (v, w) between them the size of minimal path cover doesn't change (just the path which included w is one edge shorter) example is in Figure 4.6a. But there is one

marginal case. If w is alone in a path of the length one, the size of the minimal path cover will decrease by one as it is shown in Figure 4.6b.

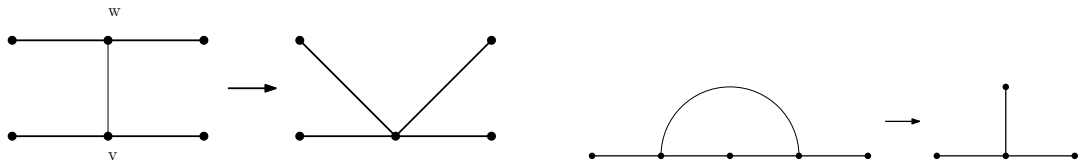


(a) $pc(G/e) = pc(G)$ (b) $pc(G/e) = pc(G) - 1$

Figure 4.6: Examples of edge contractions

Let's take two inner vertices v, w from two different paths. If we contract the edge (v, w), these two nodes become one but we can use it just once in our path cover so this operation can increase the size of the minimal path cover by one as you can see in Figure 4.7a.

Now we take both of the vertices from one path. They can be neighbours in this path and the $pc(G\backslash e) = pc(G)$ or they aren't neighbour in that path. In this case the vertices which were between the contracted nodes will be connected in a line to the new vertex thus $pc(G\backslash e) = pc(G) + 1$. Example of this situation is in Figure 4.7b.



(a) $pc(G/e) = pc(G) + 1$ (b) $pc(G - v) = pc(G) + 1$

Figure 4.7: Examples of edge contractions

The outcome is that after the edge contraction the size of the minimal path cover can be either decreased by one or increased by one or remain the same, thus

$$pc(G) - 1 \leq pc(G/e) \leq pc(G) + 1.$$

4.3 Modification effects on gap size

In Lemma 2, we proved that $pc(G) \geq sc(G)$ and we know that for cocomparability graphs the equality holds, thus $gap = 0$ for cocomparability graphs and $gap \geq 0$ for other graphs.

4.3.1 Removing an edge

Let's have a look what can happen to *gap* by an edge removal first. We want to find the worst case so we want to maximize $pc(G - e)$ and minimize $sc(G - e)$.

According to Section 4.2.1 the size of the minimum path cover can be increased by one by removing an edge.

Since the scattering number is defined as

$$sc(G) = \max\{c(G\backslash S) - |S| : S \subset V, c(G\backslash S) > 1\}$$

we denote S' the particular set S for which the maximum occurs. Let's try to make $c((G-e)\backslash S') - |S'|$ as small as possible by adding the proper edge. The cardinality of S' remains the same so we want to decrease $c((G-e)\backslash S')$ as much as possible. By removing an edge we can't decrease the number of connected components, therefore the worst case is $c((G-e)\backslash S') = c((G)\backslash S')$. This is the upper bound for the

$$c((G-e)\backslash S) - |S| : S \subset V, c((G-e)\backslash S) > 1$$

consequently for the $sc(G-e)$.

As a result, we gain that *gap* can be at most one after removing one edge from a cocomparability graph.

4.3.2 Adding an edge

We want to maximize $pc(G+e)$ and minimize $sc(G+e)$ like in the previous case.

As we know from the Section 4.2.2 the size of the minimum path cover is biggest when it remains the same, so that $pc(G+e) = pc(G)$.

We proceed in a similar way as in the previous case. However in this case we can connect at most two connected components, therefore the worst case is $c((G+e)\backslash S') = c((G)\backslash S') - 1$. Thus the upper bound for the $c((G+e)\backslash S) - |S| : S \subset V, c((G+e)\backslash S) > 1$ accordingly for the $sc(G+e)$.

In conclusion *gap* for $G+e$ is smaller or equal to one.

4.3.3 Other operations

The rest of modifications preserves cocomparability therefore all gaps remain zero.

Chapter 5

Experimental results

5.1 Generating cocomparability graphs

We want to experimentally observe certain phenomenons of behaviour of slightly changed cocomparability graphs consequently, we need to have a generator of these graphs.

We choose two parameters: n and $m \in \left(0; \frac{n(n+1)}{2}\right)$. Then we generate random graph $G = (V, E)$ where $|V| = n$ and $|E| = m$ and we number the vertices from one to n. Let's consider an order ϕ on vertices. For each $u, v \in V$ with given numbers n_1, n_2 respectively applies $u <_\phi v$ if $(u, v) \in E$ and $n_1 < n_2$. Then we make a transitive closure of ϕ and add the relevant edges to the graph G. After this modification ϕ is a strict partial order, thus G is a comparability graph. Consequently its complement $\bar{G}$ is a cocomparability graph we searched for.

Algorithm 5: Generate a cocomparability graph

Input: parameters n, m where $m \in \left(0; \frac{n(n+1)}{2}\right)$
Output: A cocomparability graph on n vertices
Generate random graph $G = (V, E)$ on n vertices with m edges
Number the vertices
foreach *vertex* v **do**
  foreach $w \in N(v)$ *where* $v < w$ **do**
    foreach $u \in N(v)$*where* $u < v$ **do**
      if $(u, w) \notin E$ **then** Add (u, w) to the graph G
    end
  end
end
return *complement of graph* G

The inconvenience of this algorithm is that we don't know the number of the graph's edges. The bigger the m is, the smaller the size of the resulting edge set is.

On the other hand, the graph construction implies that an ordering ϕ is an umbrella-free ordering.

5.2 Test summary

We are going to use (n, m) for the indication of the graph with n nodes and m parameter for generating edges. For each pair of parameters we generated a hundred of different cocomparability graphs and then applied all five our operations on the particular graph from once to ten times.

We tested graphs with 20, 40, 60, 80, 100 nodes with four different parameters for edge generation each.

Graphs with parameters $(20, 20)$, $(40, 50)$, $(40, 100)$, $(60, 50)$, $(80, 100)$, $(100, 100)$, $(100, 500)$ have too many edges. They have just one path almostr always, even after all of the modifications. All of our hundred testing graphs had a Hamiltonian path.

In the cases $(20, 40)$, $(40, 200)$, $(60, 300)$, $(60, 500)$, $(80, 500)$, $(100, 1000)$, there are still too many edges so the graphs are covered by one or too few paths. Thus the changes in the number of paths are negligible. In these graphs the situation where the minimal path cover was bigger than one occurred but the average size of the minimal path cover is still rounded to one. The rest of the instances will be much more interesting.

5.2.1 Path cover

Next follows Table 5.1 showing the changes of the path cover computed by Algorithm 4. We introduce just the one-modification operations (addition/removal of one edge, vertex, one edge contraction) in this table for easier comparison with the theory. We can see that results for operations preserving cocomparability are in exact correspondence with theoretical results from Section 4.2.

Results for the rest of operations (addition and removal of an edge) usually confirmes the theory as well, but there are few exceptions. The most numerous deviations from the theory are eight cases from hundred for parameters $(80, 1500)$. The Algorithm 4 gives us always a path cover of the graph, but just for cocomparability graphs it is proven that it is the minimal one. Generally, the problem of finding the minimal path cover is NP-complete. Thus situations where the size of a path cover is bigger than we predicted by theory can occur, but obviously it can't be smaller that the theoretical prediction.

The fact that graphs with more edges (with smaller m) hold to theoretical expectations more than graphs with less edges is quite predictable. An interesting phenomenon is that from our experiment it seems that the size of diversion from theory depends more on the size of graph than on the number of a paths in a found path cover.

In Figures 5.1, 5.2, 5.3 we provide diagrams of sizes of path covers found by Algorithm 4 and their changes while we modify the graph more than once.

We can observe quite expectedly that the more edges or vertices we remove, the bigger path cover we get, whereas the more edges or vertices we add, the smaller the size of path cover is. In the edge modifications edge additions change the size of path cover slightly more than removing edges. The steepest decrease can be seen for the removing of vertices. The cocomparability is preserved by this operation and by each addition of an universal vertex we decrease the size of the minimal path cover by one.

We can notice that there is no perceptible increase nor decrease just small haphazard diversions for an edge contraction. The theory tells us that the size of the minimal path cover can be increased or decreased by one, nevertheless often there are no changes at all.

Graph properties				Changes in number of paths				
n	m	ANOE	ANOP	$-e$	$+e$	$-v$	$+v$	c
20	60	63	2	$8 \times +1$ $2 \times +2$	$3 \times +1$ 9×-1	$16 \times +1$ 10×-1	44×-1	$3 \times +1$ 9×-1
20	80	40	3	$20 \times +1$	$8 \times +1$ 29×-1	$19 \times +1$ 16×-1	85×-1	$13 \times +1$ 13×-1
40	450	40	16	$26 \times +1$ $2 \times +2$	$12 \times +1$ 35×-1 $3 \times +2$ $1 \times +3$	$25 \times +1$ 34×-1	100×-1	$9 \times +1$ 17×-1
60	1000	67	22	$26 \times +1$ $2 \times +2$ $1 \times +3$	$17 \times +1$ 26×-1 $6 \times +2$ $1 \times +3$	$27 \times +1$ 34×-1	100×-1	$4 \times +1$ 18×-1
80	1000	327	3	$17 \times +1$ $4 \times +2$ $2 \times +3$	$12 \times +1$ 4×-1 $8 \times +2$ $4 \times +3$ $1 \times +4$	$15 \times +1$ 7×-1	82×-1	$6 \times +1$ 3×-1
80	1500	142	19	$22 \times +1$ $3 \times +2$ $2 \times +3$ $3 \times +4$	$21 \times +1$ 11×-1 $8 \times +2$ $4 \times +3$ $4 \times +4$ $2 \times +6$ $1 \times +7$ $1 \times +8$	$32 \times +1$ 28×-1	100×-1	$15 \times +1$ 21×-1
100	2500	156	27	$23 \times +1$ $3 \times +2$ $1 \times +3$ $1 \times +4$	$19 \times +1$ 20×-1 $5 \times +2$ $2 \times +3$ $3 \times +4$ $2 \times +5$ $1 \times +6$	$25 \times +1$ 29×-1	100×-1	$12 \times +1$ 14×-1

Table 5.1: Changes of a size of path cover given by the algorithm. Where ANOE and ANOP are abbreviations for average number of edges and average number of paths respectively.

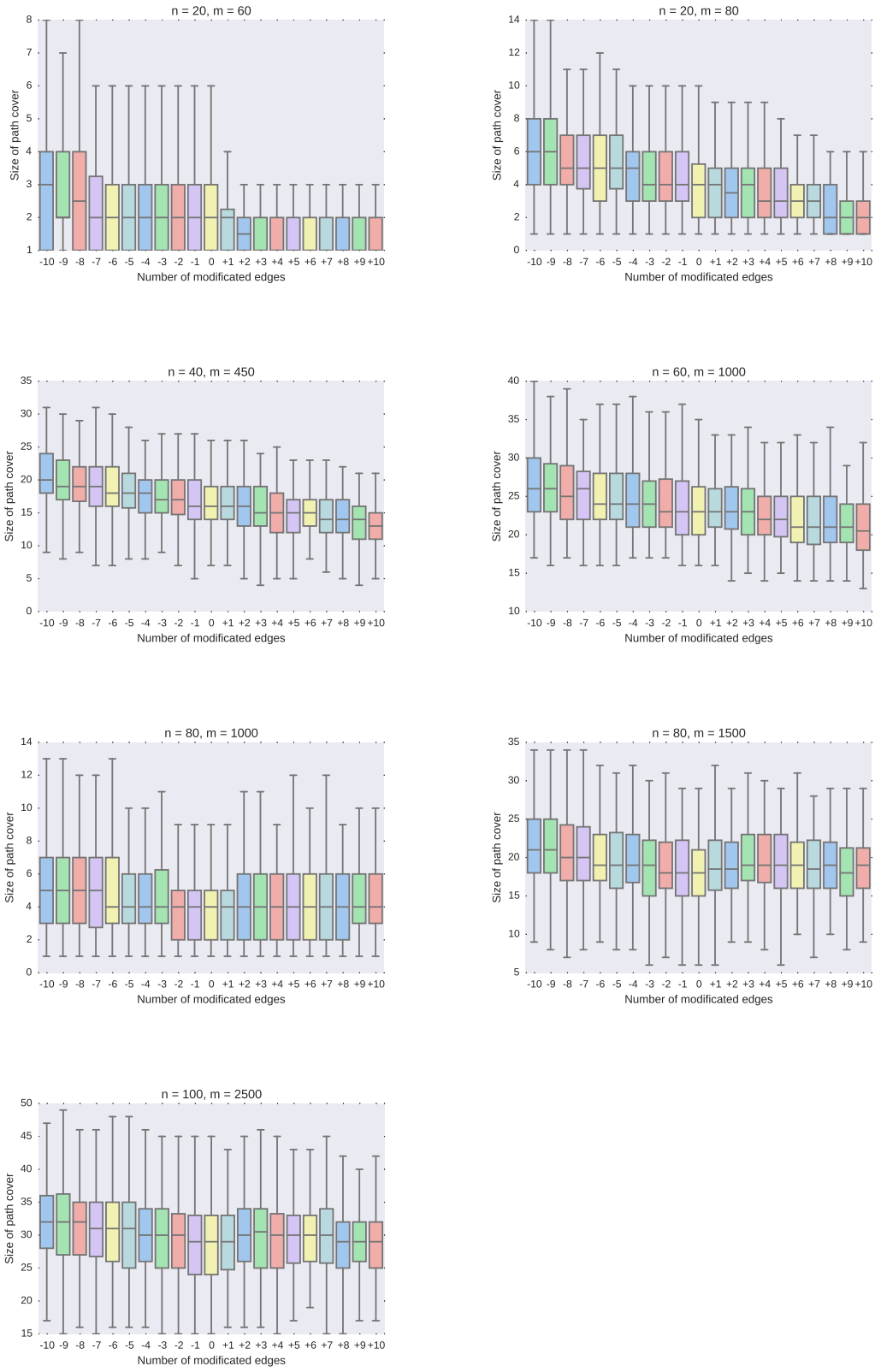


Figure 5.1: Sizes of path cover for adding/removing edges

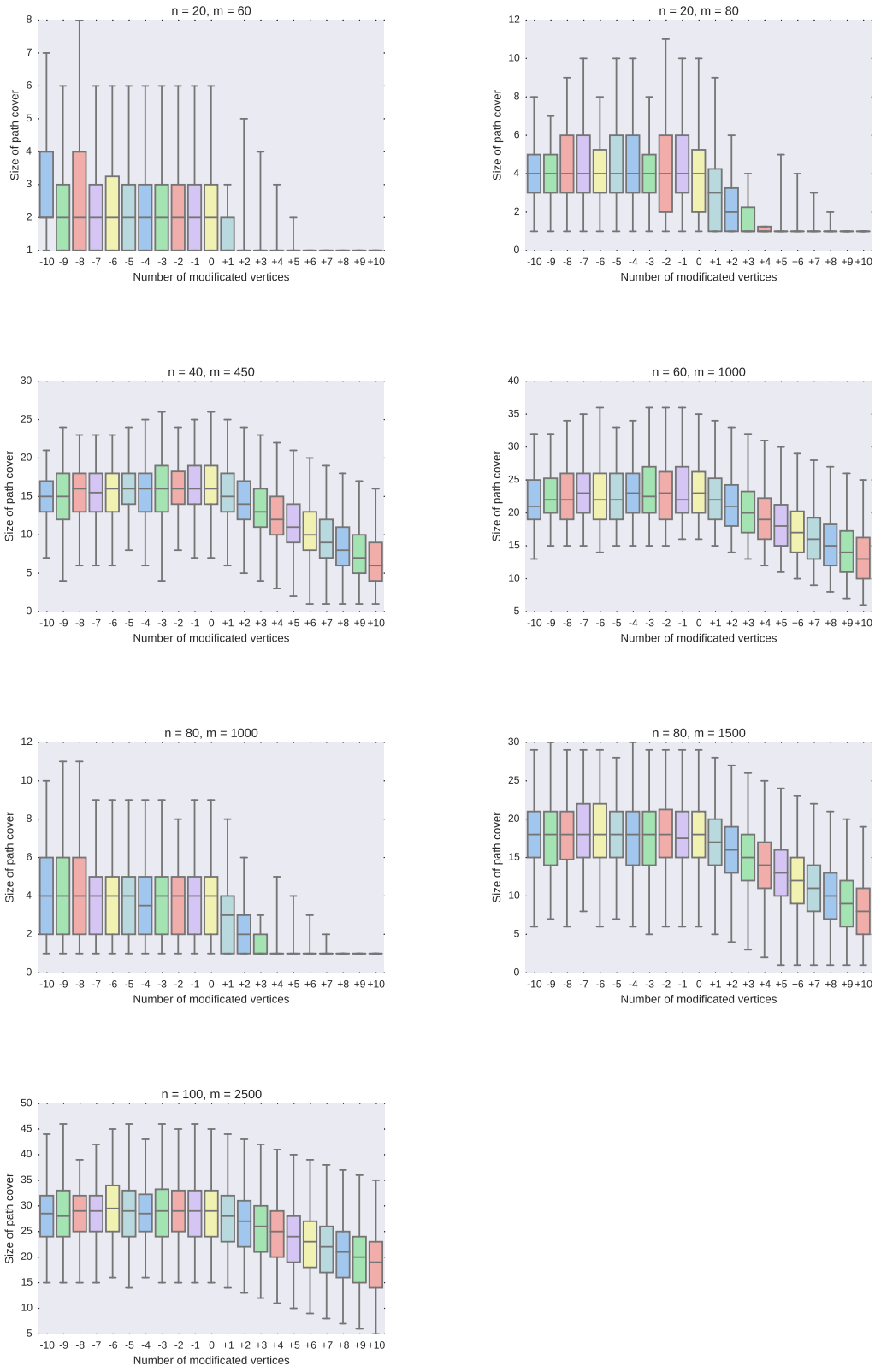


Figure 5.2: Sizes of path cover for adding/removing vertices

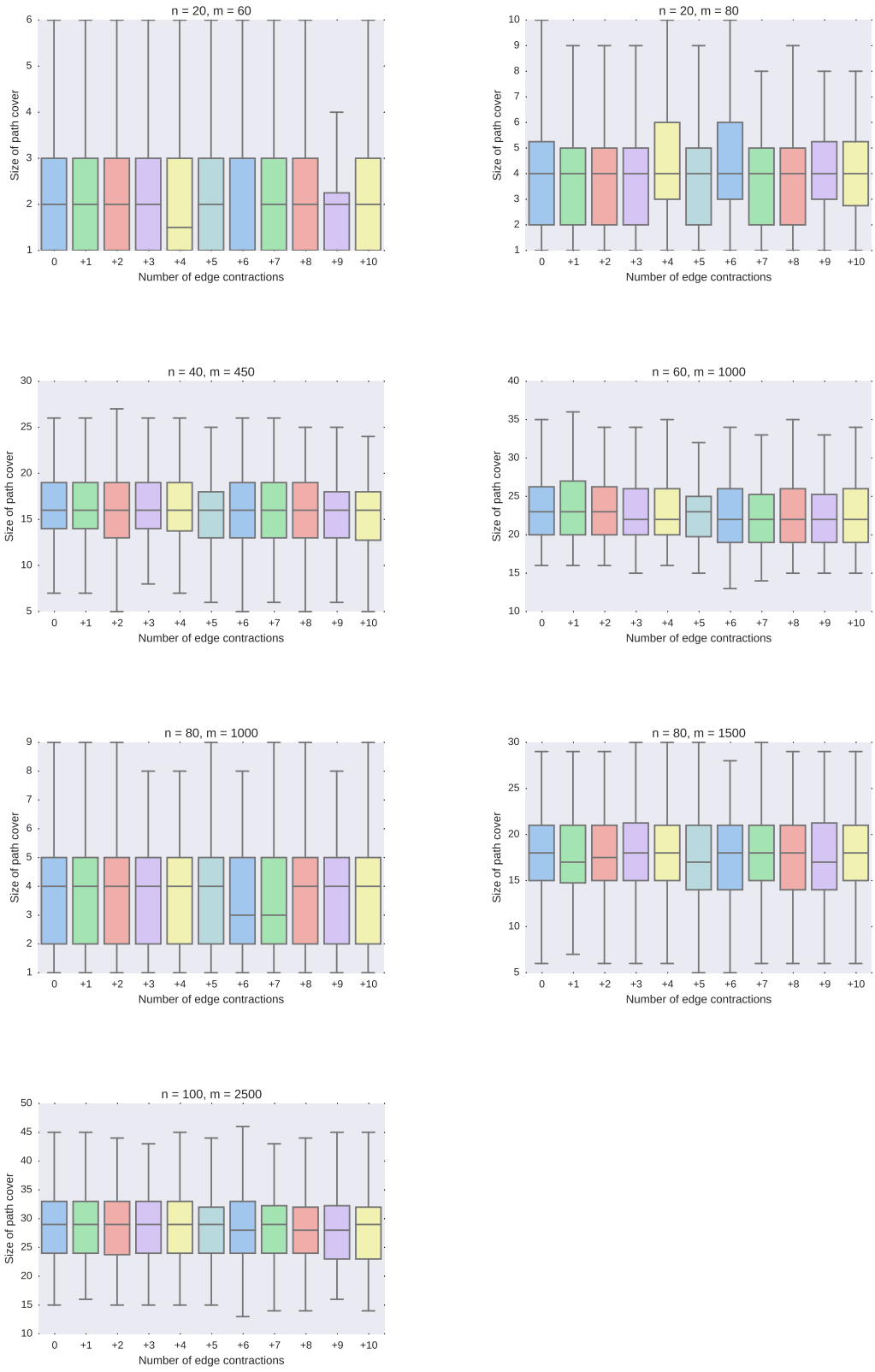


Figure 5.3: Sizes of path cover for edge contractions

5.2.2 Gaps

The answer to the question whether the algorithm provides us the right result is the gap value. If the gap is zero given path cover is the minimal one.

Thus all gaps for all cocomparability graphs are zero, so we discuss just gaps for adding and removing edges.

As you can see in Table 5.2, most of the gaps remain zero but there are gaps even bigger than one which is in contradiction with theory from Section 4.3. The reason is that we looked what can happen to gap just by modifying graph, but we didn't check process of the algorithm. There are more nonzero gaps for adding edge than removing edge. The most of the nonezero gaps for removing an edge is for $(80, 1000)$ graphs and there are 13 of them, while for adding edge there are 40 nonezero gaps for $(80, 1500)$ graphs.

In Figure 5.4 we can see that the more we modify the graph the more none-zero gaps appear. It also shows that adding edges baffle the algorithm more than removing edges.

Graph parameters		None-zero gaps	
n	m	$-e$	$+e$
20	60	2×1, 3×2, 4×4, 1×5	2×1, 4×2, 3×4, 1×5 1×6
20	80	2×1, 3×2, 3×3	10×1, 7×2, 3×3, 3×4 1×5, 1×7
40	450	2×1, 1×4	7×1, 6×2, 2×3, 1×5
60	1000	1×1, 1×2, 2×5	15×1, 4×2, 6×3, 2×4 3×6, 1×8, 1×12
80	1000	2×1, 1×2, 1×3, 1×4 2×5, 2×6, 1×7, 1×10 1×12, 1×13	5×1, 1×2, 3×3, 1×4 3×5, 2×6, 5×7, 2×8 3×9, 2×10, 1×11, 1×13 1×14, 1×19, 1×23, 1×26
80	1500	4×1, 1×2, 1×3, 1×6 1×11	5×1, 9×2, 5×3, 3×4 7×5, 2×6, 2×7, 2×8 2×9, 2×11, 1×15
100	2500	2×1, 2×2, 1×3, 1×4 1×5	10×1, 11×2, 8×3, 1×4 6×5, 4×7, 1×9, 1×10 1×11, 2×12, 2×14, 1×22

Table 5.2: Nonzero gaps for operations spoiling cocomparability

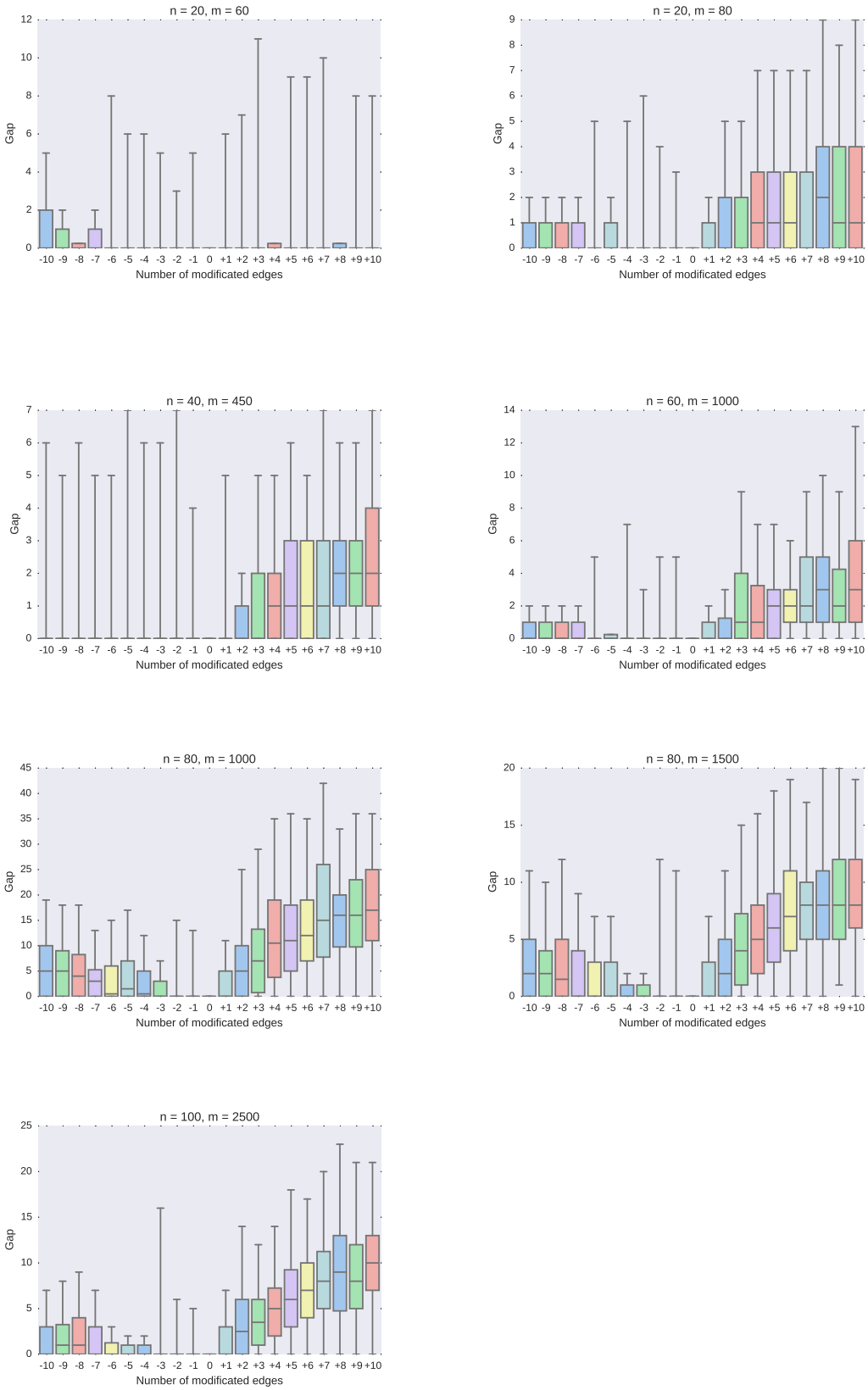


Figure 5.4: Sizes of gaps for adding/removing edges

Chapter 6

Description of the attachments on the cd

We include the sources of our program for reference and completeness with basic documentation and description. Note that the implementation itself is not a primary goal of the thesis.

There are six files on the cd:

1. Program which we used to gain the data for experimental results
2. Three partial data results
3. Text file with parameters for tested graphs
4. Text file with the description of the other text files

6.1 Program implementation

For programming we used Python 2.5 with the NetworkX library for work with graphs, see Hagberg et al. (2008).

The program is in file `mpc_algo.py` consists of five principal functions. Each of them is an implementation of an algorithm described in this work.

6.1.1 Cocomparability graph generator

Function `gen_cocomp_graph(n, m=None)` takes parameters n, m and return cocomparability graph with n vertices. It is an implementation of Algorithm 5.

6.1.2 LDFS+

Function `ldfsp(G, pi)` takes a graph G and an umbrella-free ordering π and return another ordering. It is an implementation of Algorithm 1.

6.1.3 RMN

Function `rmn(G,sigma)` takes a graph G and an ordering σ given by LDFS+ and return different ordering and a set of paths covering graph G. It is an implementation of Algorithm 2.

6.1.4 CERTIFY

Function `certify(G, tau, P)` takes a graph G, an ordering τ given by RMN and return set of vertices, separator S and disjoint subpaths of P. It is an implementation of Algorithm 3.

6.1.5 MPC

Function `mpc_cocomp(G,pi)` takes a graph G and an umbrella-free ordering π and return set of paths covering graph G, set of vertices, separator S and disjoint subpaths of P. This algorithm just put the previous algorithms together as you can see in its pseudocode in Algorithm 4.

Further comments are included in `mpc_algo.py` file.

6.2 Data files

6.2.1 Basic results

The file `basic_results.txt` contains some partial data for all tested pairs of parameters. According to this data, we decided which parameters are worth to focus on more.

For each pair of parameters it contains average number of edges and average size of the minimal path cover. Then for each operation on these graphs with one to ten iterations it contains average number of edges and average size of a path cover of graphs modeficated by the operation. It also contains the number of changes in a path cover size and number of changes in gap, both according to the graphs before modifications.

6.2.2 Path cover diagram data

The file `diagram_result_pc.txt` contains data we used for creating graphs in Figures 5.1, 5.2, 5.3.

For each pair of observed parameters it contains a list of all hundred sizes of path covers for all modifications. It is sorted as follows: First are the edge modifications starting by removing ten edges, then removing nine edges and continue through the size of path cover of the original graph to the addition of ten edges. Vertex modifications and edge contractions follow the same pattern.

6.2.3 Gap diagram data

The last data file `diagram_result_gap.txt` contains data we used for creating diagrams in Figures 5.4.

For each pair of observed parameters it contains a list of all hundred sizes of gaps for removing and adding edges. It is sorted as in the previous section: First are the edge modifications starting by removing ten edges, then removing nine edges and continue through the size of path cover of the original graph to the addition of ten edges.

6.3 Tested graphs parameters

The last file included `testovaci_grafy.txt` is an input for program `mpc_algo.py`. It contains parameters for all graphs we run in the program.

Chapter 7

Conclusion

We implemented Algorithm 4 from Corneil et al. (2013) for searching for the minimal path cover on cocomparability graphs and tested it on cocomparability graphs generated by Algorithm 5. Then we examined five graph operations and their relationship to cocomparability. Next we observed how these operations can change the size of minimal path cover of a graph. We studied how the result of the Algorithm 4 after applying our modifications.

It turned out that even on cocomparability graphs modified by operation spoiling cocomparability the output of the Algorithm 4 was usually quite accurate.

We discovered few interesting facts which deserve closer examination:

- If the difference between the results obtained from the Algorithm 4 from theoretical expectations really depends on the number of graph vertices significantly more than on the size of path cover.
- What is the impact of edge contraction on the size of the minimal path cover,
- Why there are more none-zero gaps when adding edges than when removing edges and how we can modify the algorithm to perform better in this case.

Bibliography

S. R. Arikati and C. Pandu Rangan. Linear algorithm for optimal path cover problem on interval graphs. *Inform. Process. Lett.*, 35:149–153, 1990.

B. Bollobas. *Modern graph theory*. Graduate Texts in Mathematics. Springer, New York, 1998. ISBN 978-1-4612-0619-4.

D. G. Corneil and R. Krueger. A unified view of graph searching. *SIAM J. Discrete Math.*, 22(4):1259–1276, 2008.

D. G. Corneil, J. Dusart, M. Habib, and E. Köhler. On the power of graph searching for cocomparability graphs. *Journal of Acquired Immune Deficiency Syndrome*, 49:309–319, 2008.

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

P. Damaschke. Paths in interval graphs and circular arc graphs. *Discrete Math.*, 112:49–64, 1993.

J. S. Deogun, D. Kratsch, and G Steiner. 1-tough cocomparability graphs are hamiltonian. *Discrete Mathematics*, 170(8):99–106, 1997.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

D. Kratsch and L. Stewart. Domination on cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 6(3):400–417, 1993.

D. Kratsch, T. Kloks, and H. Müller. Computing the toughness and scattering number on interval and other graphs. Technical Report 2237, INRIA, Rennes, March 1994.

G. B. Mertzios and D. G. Corneil. A simple polynomial algorithm for the longest path problem on cocomparability graphs. *SIAM J. Discrete Mathematics*, 26 (3):940–963, 2012.

List of Figures

List of Tables

List of Algorithms