UNIVERSITY OF TWENTE.

Discrete Mathematics and Mathematical Programming

Master's thesis

Graph-theoretical aspects of constraint solving in the SST project

by Jacob J. Koelewijn

Supervisors:
Dr. Georg J. Still
Prof.dr. Marc J. Uetz
Ir. Matthijs J. Bomhoff

Enschede,
August 22, 2011

Acknowledgements

This thesis is the result of my work for the Final Project of my master's degree program in Applied Mathematics at the University of Twente.

First of all I'd like to say thanks to Georg Still, my main supervisor. Without his help this thesis wouldn't be possible. He guided me through the process and assisted me where necessary, both during the research itself and the writing of this thesis.

Also thanks to Matthijs Bomhoff, who was my main adviser. He made very useful remarks about my work during discussions. Thanks to Marc Uetz, for taking part in my graduation committee.

I'm also grateful to Hans Tragter, who provided insight in the Smart Synthesis Project. Thanks to Tim, Sophie and Jasper, whom I shared an office with while working on my thesis, for the relaxed and pleasant atmosphere. Furthermore, I would like to thank all the other people working in the Citadel building for the friendly chats during coffee and lunch breaks.

Finally I'm thankful to my family and friends for supporting me while working on this project.

Contents

1 Introduction

This thesis covers some graph-theoretical aspects of constraint solving in the *Smart Synthesis Tools project* (*SST project*). So let's begin with a small introduction of the project and the problems that will be of interest for this thesis. A more general overview of the SST project can be found in Appendix A.

The aim of the SST project is to develop software with the ability to generate and analyze a number of different designs of a product or a machine. The constraint solving part of this software involves solving large underconstrained systems of equations. This problem is analyzed by constructing a bipartite graph associated to the structure of the system of equations. This will be discussed in Chapter 2.

Different decompositions of the bipartite graph and their use for solving the system of equations are investigated in Chapter 3. Also, properties of the decompositions are proved in a new way in terms of maximum matchings. In Chapter 4, the Quasi-Newton method and its use for the SST project will be described, namely to find solutions of the subsystems that are found using the decompositions.

Besides the decompositions, the bandwidth reduction problem for unsymmetric matrices is investigated in Chapter 5. A reduction of bandwidth allows for more efficient storage and calculation when solving big sparse systems of linear equations using banded algorithms. Also in this case, the problem will be approached by looking at a bipartite graph corresponding to a problem instance.

Finally, the conclusions of this thesis will be summarized in Chapter 6 together with recommendations for further research.

2 Structure of systems of equations

2.1 Introduction

Let us call the the software that is being developed for the SST project the *SST framework*. One important goal of the SST framework is to generate a number of (different) designs of a product or machine given a *model* which contains certain properties and constraints. Such a model consists of *parameters* and *rules*. Actually, in the SST framework, the parameters and rules are ordered within a tree to provide different abstraction levels, but this is outside the scope of this thesis.

The *parameters* of the model describe the properties of the final design, such as width, height, location of certain components, material, etc. A specific assignment of values to the parameters correspond to a unique design. To be able to analyze them, all parameters in this thesis will be real variables.

The *rules* of the model describe the constraints for the final design, such as a minimal and maximal width, a specific distance between two components, a direct relation between parameters, etc. In a feasible design, all rules must be satisfied. For the sake of analysis, all rules in this thesis will be (nonlinear) equations. They will be implemented as nonlinear functions of the parameters that should evaluate to zero for a valid design/solution. Requirements of the SST project dictate that the functions have a "black box" property, i.e. no analytical information about the functions is available. The only information that is available to the SST framework is which parameters are explicitly present in a rule and the evaluation of a rule-function given the values of its dependent parameters.

Because more than one design should be feasible for a specific model, the model is in fact an underdetermined system of nonlinear equations. This chapter will give the most important definitions used in this thesis in Section 2.2. An introduction on the structural analysis that will be useful for

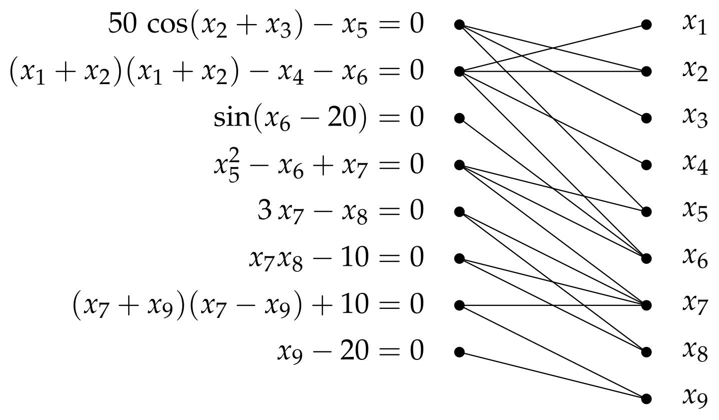


Figure 2.1: An example of a system of equations with it's associated bipartite graph

Chapter 3 can be found in Section 2.3 and Section 2.4 will give an introduction into the generation of a feasible design using an interactive solver.

2.2 Notation and definitions

2.2.1 Problem instance

We are working with a problem instance that can be defined as follows. Let $h : \mathbb{R}^n \to \mathbb{R}^p$. Then $h(x) = 0$ represents a system of p (nonlinear) equations. Because of the nature of the smart synthesis problems this system will most likely be under-constrained, i.e. $p < n$. Let

$$h_l(x) = 0, \quad l \in L = \{1, \ldots, p\}$$

be the equations of this system. p is the number of equations and x is the vector of variables x_i with $i \in I = \{1, \ldots, n\}$ where n is the number of variables.

We can construct a bipartite graph associated to this system. Let $G = (V, E)$ be this bipartite graph with vertex set $V = L \cup I$ and edge set $E \subseteq L \times I$. L and I are the vertex classes of the bipartite graph. There is an edge $e = (\ell, i) \in E$ iff $h_\ell(x)$ depends explicitly on x_i.

Figure 2.1 shows what an instance of this problem may look like.

2.2.2 Definitions

Sets

A *set* is a unordered collection of distinct objects. An object in a set is called an *element* of the set. Let A and B be two sets. The *union* $A \cup B$ is the set of all elements that are an element of either A or B. The *intersection* $A \cap B$ is the set of all elements that are an element of both A and B. If $A \cap B = \emptyset$, then A and B are said to be *disjoint*. The *difference* $A \backslash B$, is the set of all elements that are an element of A but not of B. Note that not all elements of B have to be in A. The *symmetric difference* $A \bigtriangleup B$ is the set $(A \cup B) \setminus (A \cap B)$. In other words, the symmetric difference is the set of elements that are in A or in B but not in both. The cardinality $|S|$ of a set S is the number of elements of S.

Graphs

A *simple graph* (or just *graph*) $G = (V, E)$ consists of a set V of vertices and a set E of edges. An *edge* is an unordered pair of distinct vertices of V. Two different vertices $v_1, v_2 \in V$ are called *adjacent* when there exists an edge $(v_1, v_2) \in E$. Two different edges are called *adjacent* when they share at least one vertex. A *walk* is a sequence of consecutive (adjacent) edges. A *path* is a walk with distinct edges where every vertex is traversed at most one time. A *tour* is a walk in which the first and last vertices are the same. A *cycle* is a path in which the first and last vertices are the same. Two vertices v_1 and v_2 are *connected* when there exists a path in which the first vertex is v_1 and the last vertex is v_2. A graph $G' = (V', E')$ where V' is a subset of V and E' is a subset of E containing only pairs of vertices in V' is called a *subgraph* of G. For a set of vertices $X \subseteq V$, we use $G[X]$ to denote the *induced subgraph* of G with vertex set X and with edge set $E \cap (X \times X)$. In words: the edge set of $G[X]$ is the subset of E consisting of those edges with both ends in X.

A *directed graph* $G = (V, A)$ consists of an collection V of vertices and a collection A of arcs. An *arc* is an ordered pair of distinct vertices of V. An arc (x, y) with $x, y \in V$ is directed from x to y. A *walk* in a directed graph is a sequence of consecutive arcs following the direction of the arcs. A directed graph is *strongly connected* if for for every two vertices $v_1, v_2 \in V$ there exists a walk form v_1 to v_2 and a walk from v_2 and v_1. The *strongly connected components* of a directed graph are the maximal strongly connected subgraphs. For a set of vertices $X \subseteq V$, we use $G[X]$ to denote the *induced subgraph* of G with vertex set X and with arc set $A \cap (X \times X)$.

A *bipartite graph* $G = (V, E)$ is a graph that has two vertex classes L and I such that $L \cup I = V$ and $L \cap I = \emptyset$. A *vertex class* of G is a vertex set $L \subseteq V$ with the property that there is no edge (ℓ_1, ℓ_2) in E with $\ell_1, \ell_2 \in L$.

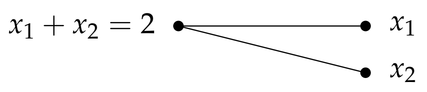
For ease of notation consider a set operation between a graph $G = (V, E)$ and an edge set E' as an operation between E and E', i.e. $G \cap E'$ can be read as $E \cap E'$. Similarly consider a set operation between a graph $G = (V, E)$ and an vertex set V' as an operation between V and V'. In case of ambiguity $E(G)$ will be used to denote the edge set E of G and $V(G)$ will be used to denote the vertex set V of G.

Matchings

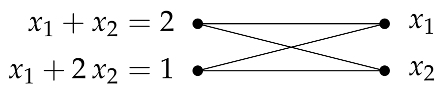
Given a bipartite graph G as defined in Section 2.2.1, a *matching* for G is a subset $M \subseteq E$ such that every vertex of G is incident to at most one edge of M. A *maximal matching* of a bipartite graph G is a matching M that is not a proper subset of any other matching in G. A *maximum matching* of a bipartite graph G is a matching M with the property that there exists no other matching M' of G with $|M'| > |M|$. A matching M *covers* a vertex $v_1 \in V$ when there exists a vertex $v_2 \in V$ with $(v_1, v_2) \in M$. A matching M *covers* a set of vertices $V' \subseteq V$ when M covers all vertices in V'. A *perfect matching* of a bipartite graph G is a matching M with the property that M covers V. Note that this is only possible when $|L| = |I|$.

2.3 Consistency concept

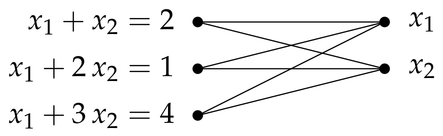
To give an idea of consistency consider the following three systems of equations:



(2.1)



(2.2)



(2.3)

System (2.1) has an infinite number of solutions. For every value of x_1 there is a value $x_2 = 2 - x_1$ that makes the equation sound. In this system there is one equation and there are two variables.

System (2.2) has exactly one solution ($x_1 = 3$, $x_2 = -1$). The system has two equations and two variables.

System (2.3) has no solution at all. The reason for this is that the first two equations only allow x_1 and x_2 to have values 3 and -1 respectively. The third equation contradicts to this.

Most of the time a system of equations does not have a solution when it has more equations than variables. Moreover, when a subsystem has more equations than variables, the whole system has no solution most of the time. This is a situation we want to avoid. We need a definition.

Definition 2.1. Let $G = (V, E)$, with $V = L \cup I$, be a bipartite graph. For a subset $L_0 \subseteq L$, the *neighbor set* $N(L_0) \subseteq I$ is the set of nodes in I that are adjacent to at least one node in L_0.

Definition 2.2. A system of equations and its corresponding bipartite graph are called (structurally) *consistent* when the following holds:

$$|N(L_0)| \geq |L_0| \quad \forall L_0 \subseteq L. \tag{2.4}$$

When a system is consistent, a situation like (2.3) is automatically avoided. There is another criterium for consistency.

Corollary 2.3. *A system of equations is (structurally) consistent iff its associated bipartite graph contains a (maximum) matching covering L.*

Proof. This theorem is a direct consequence of the definition of consistency and Hall's theorem, see Hall (1935). $\square$

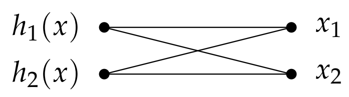


Figure 2.2: Bipartite graph corresponding to the Systems 2.5, 2.6 and 2.7

Note that for a system of equations where the number of equations is the same as the number of variables, as a consequence of Corollary (2.3), consistency implies that its associated bipartite graph contains a perfect matching.

Let us call a system h of equations $h_\ell(x) = 0$ (structurally) *under-constrained* when its associated bipartite graph has a maximum matching covering all nodes of L and $|L| < |I|$, i.e. there are more variables than equations. A system h of equations $h_\ell(x) = 0$ is (structurally) *over-constrained* when its associated bipartite graph has a maximum matching covering all nodes of I and $|L| > |I|$, i.e. there are more equations than variables. A system h of equations $h_\ell(x) = 0$ is (structurally) *well-constrained* when its associated bipartite graph has a perfect matching. A bipartite graph is called under-, over- or well-constrained when its associated system is respectively under-, over- or well-constrained.

2.3.1 Limitations and possibilities

Let's have a look at the limitations of analyzing a system of equation using the structure of its associated bipartite graph. As Ait-Aoudia et al. (1993) pointed out there is no one-to-one relation between a bipartite graph and the fact whether the corresponding system can be solved. For example the systems

$$\left\{ \begin{aligned} x_1 + x_2 &= 2 \\ 2\,x_1 + 2\,x_2 &= 4 \end{aligned} \right\} \tag{2.5}$$

$$\left\{ \begin{aligned} x_1 + x_2 &= 2 \\ 2\,x_1 + 2\,x_2 &= 3 \end{aligned} \right\} \tag{2.6}$$

$$\left\{ \begin{aligned} x_1 + x_2 &= 2 \\ x_1 + 2\,x_2 &= 3 \end{aligned} \right\} \tag{2.7}$$

all have the same corresponding bipartite graph (Figure 2.2). This system is (structurally) consistent, but System (2.5) has an infinite number of solutions, System (2.6) has no solutions at all and System (2.7) has just one solution. The reason for this is that the *Jacobian matrix* J_h of Systems (2.5) and (2.6) is singular.

This simple example shows clearly that it is not possible to say something about the solution space of a specific system of equations by analyzing its corresponding bipartite graph (structure).

However, it is possible to say something about the class of systems of equations that are consistent. This problem has been more thoroughly investigated by Still et al. (2010). Some of the main results of this work will be summarized in the remainder of this subsection without proof.

Let us consider a system of nonlinear equations $h : \mathbb{R}^n \to \mathbb{R}^n$ with n equations and n variables and its associated bipartite graph $G = (V, E)$ with $V = L \cup I$. It is well-known that for any solution $\bar{x}$ of $h(x) = 0$ the Newton iteration $x_{k+1} = x_k - [J_h(x_k)]^{-1} h(x_k)$ is locally quadratically convergent to $\bar{x}$ if the regularity condition holds:

$$J_h(\bar{x}) \quad \text{is nonsingular} . \tag{2.8}$$

Call *h regular* when (2.8) holds for all solutions $\bar{x}$ of $h(\bar{x}) = 0$ and *irregular* otherwise.

Corresponding to the bipartite graph $G = (V, E)$ with $V = L \cup I$ we define the function set S_G,

$$S_G = \Big\{ h : \mathbb{R}^n \to \mathbb{R}^n, h \in C^1 | h_i \text{ depends on } x_j \text{ only if } \ (i, j) \in E \Big\}$$

where $S_G \subset C^1(\mathbb{R}^n, \mathbb{R}^n)$ is endowed with the so-called *strong topology* as defined in Jongen et al. (2000).

Theorem 2.4. *When G is consistent,*

(a) and $h \in S_G$ is regular, any (sufficiently) small perturbation $\tilde{h} \in S_G$ of h will result in a regular function $\tilde{h}$.

(b) and $h \in S_G$ is irregular, by an arbitrarily small perturbation a regular function $\tilde{h} \in S_G$ can be obtained.

When G is not consistent,

(c) $h \in S_G$ is irregular. Even more, every solution $\bar{x}$ of $h(\bar{x}) = 0$ will not satisfy (2.8).

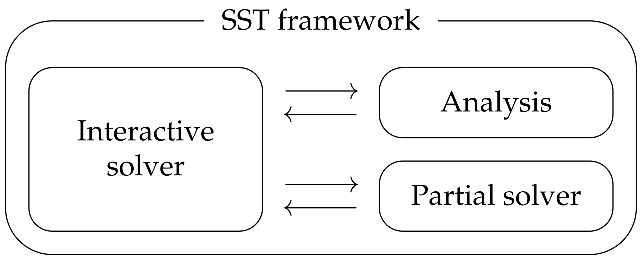


Figure 2.3: Schematic overview of interaction of the solver

(d) and $h \in S_G$, any (sufficiently) small perturbation of h will result in a function $\tilde{h} \in S_G$ such that $\tilde{h}(\bar{x}) = 0$ has no solution.

So to be able to solve h with a Newton method, it is important that G is consistent. Note that Systems (2.5) and (2.6) can be made regular by changing one of their coefficients by a arbitrarily small value $\epsilon \neq 0$, resulting in systems with just one solution. Also note that for nonlinear systems of equations, consistency does not guarantee the existence of a solution.

2.4 Interactive solver

Parallel to the investigation of the structure of the system of equations, the application to the interactive solver of the SST project will be discussed. The *interactive solver* is a part of the SST framework that generates one feasible design (solution) given a model (system of equations) as input. It will also be referred to as the *solver*. One of the requirements of the solver is that when it generates different feasible solutions, these solutions should be as different from each other as possible, i.e. they should be a good representation of the total solution space.

The *input* of the interactive solver is an (underdetermined) system of equations and its associated bipartite graph. The *user* is the decision maker that controls the interactive solver. In this thesis, the SST framework is the user, i.e. the solving process will run automatically.

The solver is interactive in the sense that it interacts with two other components of the SST framework, an analysis component (which will be covered in Section 3.3) and a partial solver (see Section 4.2). Figure 2.3 shows a schematic overview of this interaction.

First the solver receives an input from the SST framework. The solver then uses the analysis component to determine what options are available. After that, the solver uses one of these options to assign a value to one or more parameters (it might need the partial solver for that). The analysis and assigning will repeat until a (feasible) solution will be found.

3 Dulmage-Mendelsohn decomposition

There exists a unique decomposition of a bipartite graph that splits the graph in an under-, over- and well-constrained part. This decomposition was first described by Dulmage and Mendelsohn (1958, 1959, 1967). Their decomposition also decomposes the well-constrained part into irreducible components. Several authors distinguished between the two levels of the decomposition: just like Pothen (1984) this work will use the term *coarse decomposition* for decomposition in an under-, over- and well-constrained part and *fine decomposition* for the decomposition of the well-constrained part. It may be noted that the terminology of this work differs from that of Dulmage and Mendelsohn (1958, 1959, 1967) and is more similar to Pothen (1984) and Lovász and Plummer (1986).

The main goal of Subsections 3.1 and 3.2 is to provide a proof of the decomposition of Dulmage and Mendelsohn in terms of maximum matchings. This exact way of proving the Dulmage-Mendelsohn decomposition is new. However, the proof contains elements of the proofs of Pothen (1984) and Lovász and Plummer (1986).

3.1 The coarse decomposition

Lovász and Plummer (1986) give the following definition of the coarse decomposition.

Definition 3.1. The *coarse decomposition* of a bipartite graph $G = (V, E)$ with $V = L \cup I$ (see Section 2.2.1) consists of three disjoint vertex sets V_1, V_2, and V_3. Let $D \subseteq V$ be the set of all vertices of G for which there exists at least one maximum matching of G that doesn't cover that vertex. Let

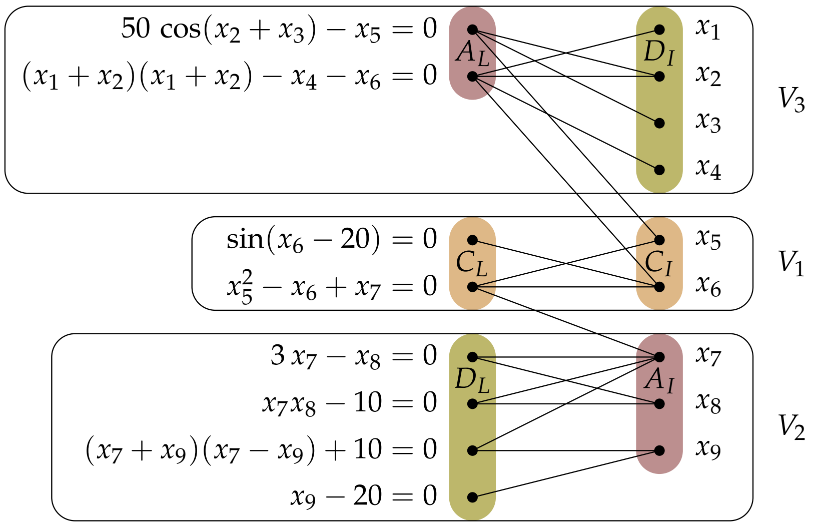


Figure 3.1: The groups V_1, V_2 and V_3

$A \subseteq V - D$ be the set of all vertices outside D that are adjacent to a vertex in D. And let $C = V - D - A$ be the set of remaining vertices. Now split the sets according to the bipartition of G, i.e. $A_L = A \cap L$, $A_I = A \cap I$, $C_L = C \cap L$, $C_I = C \cap I$, $D_L = D \cap L$ and $D_I = D \cap I$. Finally let $V_1 = C_L \cup C_I$, $V_2 = D_L \cup A_I$ and $V_3 = A_L \cup D_I$ (as in Figure 3.1).

The coarse decomposition has some useful properties that will be stated in the next theorem.

Theorem 3.2. *The coarse decomposition satisfies the following conditions.*

(a) The decomposition is unique.

(b) The are no connections between C_L and D_I and there are no connections between C_I and D_L.

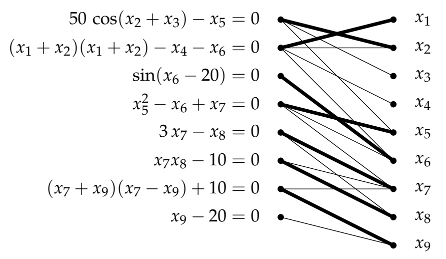
(c) There are no connections between D_L and D_I.

(d) The system corresponding to V_1 is well-constrained, the system of V_2 is over-constrained and the system of V_3 is under-constrained.

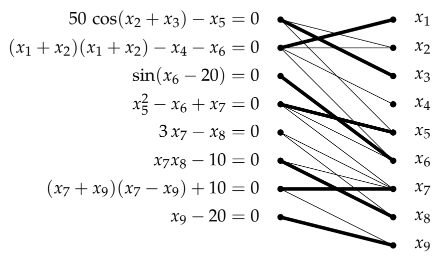
To prove Theorem 3.2 more work is needed. See the next subsection.

Proof of Theorem 3.2

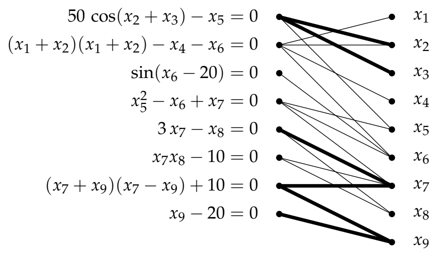
The proof is constructive and makes use of an arbitrary maximum matching M of G. It directly leads to an (polynomial-time) algorithm for the



(a) The matching M_1 consisting of the thick lines



(b) The matching M_2 consisting of the thick lines



(c) The symmetric difference $M_1 \triangle M_2$

Figure 3.2: An example of a symmetric difference

computation of the coarse decomposition.

For (a) observe that the sets are well-defined thus the decomposition is unique for a specific bipartite graph. (b) follows directly from Definition 3.1.

We are going to analyze which vertices are in D_L and D_I. It is obvious that the vertices in L not covered by M are in D_L and the vertices in I not covered by M are in D_I. We need an auxiliary lemma.

Lemma 3.3. *The following equivalences hold for vertices in D_L and D_I.*

(a) A vertex $u_L \in L$ is in $D_L \iff u_L$ is reachable by an M-alternating path starting from a vertex in L not covered by M. Clearly, this path must have an even length (zero length is also possible).

(b) A vertex $u_I \in I$ is in $D_I \iff u_I$ is reachable by an M-alternating path starting from a vertex in I not covered by M. Clearly, this path must have an even length (zero length is also possible).

Proof. Only (a) will be proven, the proof of (b) is similar by symmetry.

$\Leftarrow$ The symmetric difference of the M-alternating path and M results in a maximum matching where u_L is not covered, so $u_L \in D_L$.

$\Rightarrow$ Suppose there is a vertex $u_L \in D_L$ that is covered by M. Then there exists (by definition of D) a maximum matching M' where u_L is not covered. Now take the symmetric difference S of M and M' (see Figure 3.2 for an example with u_L corresponding to $3\,x_7 - x_8$). Because each matching can contribute maximum 1 degree to a vertex in S the symmetric difference consists only of cycles and paths. The paths are of even length because both matchings are maximum (since an odd length path would result in an augmenting path in either M or M'). Because u_L is only covered by M, u_L is an end-vertex of a path P in the symmetric difference S. P is an M-alternating path of even length so the other end-vertex of path P must be a vertex not covered by M. □

Now (c) can be proven.

Proof of (c). Suppose there is an edge (u_L, u_I) with $u_L \in D_L$ and $u_I \in D_I$. Then by Lemma 3.3 there exists an M-alternating path P_1 starting in a vertex in L not covered by M to u_L. By symmetry there exists an M-alternating path P_2 starting in a vertex in I not covered by M to u_I. If $(u_L, u_I) \notin M$ then combining P_1, (u_L, u_I) and P_2 results in an augmenting path for M, which is a contradiction because M is maximum. Notice that there can't be an edge e that is in both P_1 and P_2 because otherwise it would be possible to construct an augmenting path for M, which is a contradiction.

So let's consider the case that $(u_L, u_I) \in M$. We know there exists an M-alternating path P starting in a vertex in L not covered by M to u_L. The last edge of the M-alternating path must be in M because the first (starting) edge of P is not in M and P has even length. This means that (u_L, u_I) must the last edge of P, because otherwise (u_L, u_I) can't be in M, which is a contradiction. By the same argument there exists an M-alternating path P' starting in a vertex in I not covered by M to u_I where the last edge of P' is (u_L, u_I). But now combining the M-alternating paths P and $P' \setminus (u_L, u_I)$ results in an augmenting path for M, which is a contradiction because M is maximum. □

Before proving (d) we need two other auxiliary lemmas.

Lemma 3.4. *The following equivalences hold for vertices in A_I and A_L.*

(a) *A vertex $u_I \in I$ is in $A_I \iff u_I$ is reachable by an M-alternating path starting from a vertex in L not covered by M. Clearly, this path must have an odd length.*

(b) *A vertex $u_L \in L$ is in $A_L \iff u_L$ is reachable by an M-alternating path starting from a vertex in I not covered by M. Clearly, this path must have an odd length.*

Proof. We'll only prove (a), the proof of (b) is similar by symmetry.

$\Leftarrow$ Let v_L be the vertex before u_I in the M-alternating path. By Lemma 3.3 we know that v_L is in D_L. Obviously u_I is adjacent to v_L so u_I is in A (by (c) u_L cannot be in D_I). Because G is bipartite we know u_I is in A_I.

$\Rightarrow$ Because $u_I \in A_I$ there exists a vertex $v_L \in D_L$ adjacent to u_I. By Lemma 3.3 it is known that there exists an M-alternating path P from a vertex in L not covered by matching M to vertex v_L of even length. If P goes through u_I we are done. So suppose that P does not go through u_I. Note that in this case v_L can't be matched to u_I by M because the last edge of P must be a matching edge. Now $P \cup (v_L, u_I)$ is an M-alternating path of odd length. □

Lemma 3.5. *For an edge $(\ell, i) \in M$ with $i \in A_I$ it is always true that $\ell \in D_L$. And for an edge $(\ell, i) \in M$ with $\ell \in A_L$ it is always true that $i \in D_I$.*

Proof. Only the first statement first will be proven, the second statement is true by symmetry. Lemma 3.4 tells us that there exists an M-alternating path P starting in a vertex in L not covered by M to i. P has odd length and its first edge is not in M because the starting vertex of P is not covered by M. So the last edge of P is also not in M. Combining P with (i, ℓ) would result in an M-alternating path from a vertex in L not covered by M to ℓ of even length. Now by Lemma 3.3 we know that $\ell \in D_L$. □

Finally (d) can be proven.

Proof of (d). Let us begin with the proof of the statement that the system corresponding to V_2 is over-constrained. Because $V_2 = D_L \cup A_I$ we know by Lemmas 3.3 and 3.4 that V_2 consists of all vertices that are reachable by an M-alternating path starting from a vertex in L not covered by M.

Lemma 3.5 implies that when a vertex is in A_I, the vertex that is matched to it by M is in D_L. Further, all vertices in A_I are covered by M. So $M \cap (D_L \times A_I)$ is a maximum matching for $G[V_2]$ covering A_I. D_L consists of all vertices matched to a vertex in A_I and all vertices in L not covered by M. Thus, if $V_2 \neq \emptyset$, then $|D_L| > |A_I|$. Consequently $G[V_2]$, and thus the system corresponding to V_2, is over-constrained.

By symmetry this also means that the system corresponding to V_3 is under-constrained.

To prove that the system corresponding to V_1 is well-constrained consider matching M. By Theorem (b) and Lemma 3.5 we know that vertices in C ($C_L \cup C_I$) can only be matched by M to vertices in C. Because every vertex in C is covered by M by definition every vertex in V_1 ($= C$) is matched to another vertex in V_1 by M. This means that $M \cap E(G[V_1])$ is a maximum matching for $G[V_1]$ and $|C_L| = |C_I|$ with the result that the system corresponding to V_1 is well-constrained. □

3.1.1 Implementation

To construct a coarse decomposition Ait-Aoudia et al. (1993) provided an efficient algorithm. See Algorithm 3.1.

Algorithm 3.1 An algorithm for the coarse decomposition

```
Input: A bipartite graph $G = (V, E)$ with $V = L \cup I$
Output: Three vertex sets $V_1$, $V_2$ and $V_3$ with $V_1 \cup V_2 \cup V_3 = V$
1 find a maximum matching $M$ of $G$
2 directed graph $G' \leftarrow (V, \emptyset)$
3 foreach edges $(\ell, i)$ in $G$ do
4     add arc $(\ell, i)$ to $G'$
5 foreach edges $(\ell, i)$ in $M$ do
6     add arc $(i, \ell)$ to $G'$
7 vertex set $V_2 \leftarrow$ all descendants of sources of $G'$
8 vertex set $V_3 \leftarrow$ all ancestors of sinks of $G'$
9 vertex set $V_1 \leftarrow V - V_2 - V_3$
```

This algorithm follows directly from Lemmas 3.3 and 3.4. It is easy to see that when there is a (directed) walk in the directed graph G' between a vertex v_1 and a vertex v_2 there exists an M-alternating path in G between v_1 and v_2.

Lines 2 to 9 run in $O(n+m)$ time where n is the number of vertices and m is the number of edges in G. The complexity of the whole algorithm is determined by the complexity of finding a maximum matching in line 1. This can be done in $O(m\sqrt{n})$ time using the bipartite matching algorithm of Hopcroft and Karp (1973).

3.2 The fine decomposition

The coarse decomposition results in V_1, V_2 and V_3. These are respectively the well-, over- and under-constrained parts of the bipartite graph. It is possible to decompose the well-constrained part V_1 (Figure 3.3) into even smaller parts. These parts will be called the irreducible components. As explained earlier this decomposition will be called the fine decomposition and is also due to Dulmage and Mendelsohn (1958, 1959, 1967). The notation will be similar to the notation used by Ait-Aoudia et al. (1993). But first a definition is needed.

Definition 3.6. A well-constrained bipartite graph $G=(V,E)$ with $V=L\cup I$ is called *irreducible* if every edge $(\ell,i)\in E$ is part of at least one perfect matching of G.

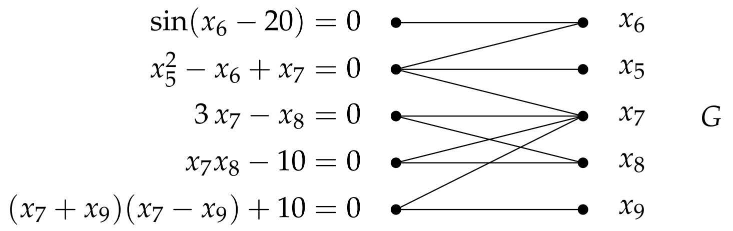


Figure 3.3: A valid instance for the fine decomposition

Definition 3.7. The *fine decomposition* of a well-constrained bipartite graph $G=(V,E)$ with $V=L\cup I$ (see Section 2.2.1) consists of q disjoint subgraphs $H_1,\dots,H_q$ constructed as follows:

Define a graph H given by G with all edges removed that never appear in a perfect matching, i.e.

$$H = (V, \{e \in E : e \text{ is in at least one perfect matching of } G\}) .$$

Now let $H_1, \ldots, H_q$ be the q components of H. Obviously, every component of H is irreducible. That is why $H_1, \ldots, H_q$ are also called the *irreducible components*.

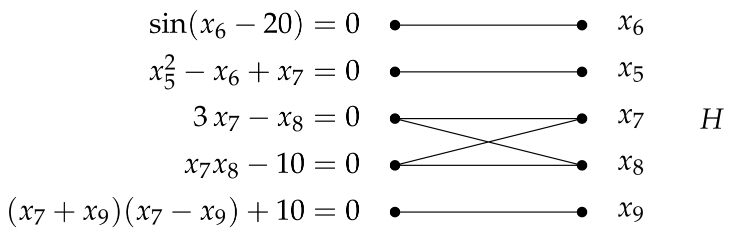


Figure 3.4: Example of H constructed from the instance of Figure 3.3

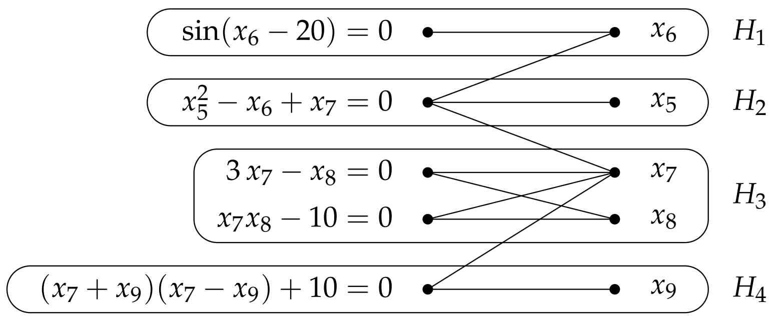


Figure 3.5: Example of a fine decomposition

An example of H and the decomposition can be found in Figures 3.4 and 3.5 respectively. The fine decomposition has some useful properties that will be stated in a theorem.

Theorem 3.8. *The fine decomposition satisfies the following conditions.*

(a) The decomposition is unique.

(b) Every subgraph $H_j \in \{H_1, \ldots, H_q\}$ is well-constrained.

(c) The subgraphs $H_1, \ldots, H_q$ can be ordered (and renumbered) in such a way that for every edge (ℓ, i) with $\ell \in H_j \cap L$, $i \in H_k \cap I$ it holds that $j \geq k$.

In words, (c) states that the subsystems corresponding to $H_1, \ldots, H_q$ can be ordered in such a way that every equation corresponding to $\ell \in H_j \cap L$

only contains variables from $\{i : i \in (H_1 \cup \cdots \cup H_j) \cap I\}$. A consequence is that this order is also an order of resolution of the associated subsystems of $H_1, \ldots, H_q$. An ordering of our example system in Figure 3.5 could be H_3, H_4, H_1, H_2.

To prove Theorem 3.8 more work is needed. See the next subsection.

Proof of Theorem 3.8

The proof is constructive and makes use of an arbitrary maximum matching M of G. It directly leads to an (polynomial) algorithm for the computation of the fine decomposition.

Proof of (a) and (b). For (a) observe that the sets are well-defined thus the decomposition is unique for a specific bipartite graph.

(b) can be proven by contradiction. We know that G has a perfect matching M. By definition M is also a perfect matching of H. Suppose now that a subgraph H_j is not well-constrained. Then H_j doesn't have a perfect matching. Because H_j is a component of H that would mean that H doesn't have a perfect matching, which is a contradiction. □

It would be useful to have a way to determine whether two vertices are in the same subgraph H_j. But first two auxiliary lemmas are needed. In these lemmas, M is a maximum (perfect) matching of G.

Lemma 3.9. *The following statements are true.*

(a) For every M-alternating tour T in G there exist(s) $m \geq 1$ M-alternating cycle(s) $C_1, \ldots, C_m$ in G for which $C_1 \cup \cdots \cup C_m$ is connected and for which it is true that every edge that appears in T also appears in $C_1 \cup \cdots \cup C_m$.

(b) For an arbitrary set of $m \geq 1$ M-alternating cycle(s) $C_1, \ldots, C_m$ in G for which $C_1 \cup \cdots \cup C_m$ is connected there exists an M-alternating tour T in G for which it is true that every edge that appears in $C_1 \cup \cdots \cup C_m$ also appears in T.

Proof. Let (a) be proven first. An M-alternating tour T for which there is a vertex v that is walked by $n \geq 2$ times in tour T can be split up in two

M-alternating tours T_1 where where v is walked by 1 time and T_2 where v is walked by $n-1$ times by the following procedure.

Let T_1 be the tour from v following T until v is reached again. Let the rest of the tour be T_2.

It is obvious that both T_1 and T_2 are M-alternating tours in G and every vertex in T is in $T_1 \cup T_2$. T_1 and T_2 can be split up by the same procedure if they contain vertices that are walked by a multiple number of times. Doing this over and over again results in the desired cycles $C_1 \cup \cdots \cup C_m$.

For (b), first sort and renumber $C_1 \cup \cdots \cup C_m$ such that $C_1 \cup \cdots \cup C_i$ is connected to C_{i+1} for $i = 1, \ldots, m-1$. Let $T_1 = C_1$. Now proceed iteratively as follows for $i = 1, \ldots, m-1$.

Let T_i be an M-alternating tour in G and let v_i be a vertex with $v_i \in T_i$ and $v_i \in C_{i+1}$. Define T_{i+1} as a tour starting in v_i, then start the tour by walking all the way through T_i (starting with an edge in M) and then continue the tour by walking all the way through C_{i+1} (again starting with an edge in M). It is obvious that T_{i+1} is an M-alternating tour in G.

Now T_m is an M-alternating tour in G containing all vertices from $C_1 \cup \cdots \cup C_m$. $\square$

Lemma 3.10. *If there exists an M-alternating cycle C containing two vertices $v_1, v_2 \in V$ then v_1 and v_2 are both an element of H_j for a specific j.*

Proof. Let M' be the symmetric difference $M \triangle C$. Now M' is also a perfect matching. By definition all edges of $C = M \triangle M'$ are in H (each edge of C is either in M or M'). Because C is a cycle it is obvious that v_1 and v_2 are connected and lie in the same component of H. $\square$

Lemma 3.11. *Two vertices $v_1, v_2 \in V, v_1 \neq v_2$ are both an element of H_j for a specific $j \iff v_1$ is matched to v_2 by M or there exists an M-alternating tour T containing both v_1 and v_2 (where T obviously has even length).*

Proof. $\Leftarrow$ If v_1 is matched to v_2 by M than obviously v_1 and v_2 are connected in H and lie in the same component of H.

So consider the case that there exists an M-alternating tour T containing both v_1 and v_2. By Lemma 3.9 we know that there exist $m \geq 1$ M-alternating cycle(s) $C_1 \cup \cdots \cup C_m$ for which every vertex in T is in $C_1 \cup \cdots \cup C_m$ and

$C_1 \cup \cdots \cup C_m$ is connected. Lemma 3.10 implies that all vertices in an M-alternating cycle are connected in H. That results in the fact that all vertices in $C_1 \cup \cdots \cup C_m$, and thus in T, are connected in H.

$\Rightarrow$ It is known that v_1 and v_2 are connected in H so there exists a path P where v_1 is the first vertex of P and v_2 the last vertex. Let $P' = \{e_1, \ldots, e_m\}$ be P with all edges in M removed. Now, for $i = 1, \ldots, m$, let M_i be a perfect matching containing e_i. For all $i = 1, \ldots, m$ take the symmetric difference $S_i = M \triangle M_i$. Because every vertex in S_i can only have degree 0 or 2 the symmetric difference consists only of cycles. Let C_i be the cycle of S_i containing e_i. Notice that C_i is M-alternating and that the edges in M adjacent to e_i are also in C_i. Because we know that either e_i and e_{i+1} are connected or there is an edge in M adjacent to e_i and e_{i+1} it follows that C_i is connected to C_{i+1} for all $i = 1, \ldots, m-1$. By Lemma 3.9 it is known that there exists an M-alternating tour T containing both v_1 and v_2. □

Now it's possible to prove condition (c) of Theorem 3.8.

Proof of (c). Suppose an ordering as stated in (c) would not be possible. The only way to achieve that is to have a "circular" sequence $H'_1, \ldots, H'_m, H'_{m+1} = H'_1$ with $m > 1$ and $H'_1, \ldots, H'_m$ distinct such that there exists at least one edge (i_j, ℓ_{j+1}) with $i_j \in H'_j \cap I$ and $\ell_{j+1} \in H'_{j+1} \cap L$ for every $j \in \{1, \ldots, m\}$. So suppose we have such a sequence. Consider an arbitrary edge (i_j, ℓ_{j+1}). We know that (i_j, ℓ_{j+1}) is not matched by M or else i_j and ℓ_{j+1} would be in the same H_j (by Lemma 3.11), which is a contradiction. Now look at an edge (i_{j+1}, ℓ_{j+2}). By Lemma 3.11 it is known that there exists an M-alternating tour T between i_{j+1} and ℓ_{j+1}. That implies there must exist an M-alternating path P_{j+1} between i_{j+1} and ℓ_{j+1} in H'_{j+1} where both the first and last edge are in M. With this it is possible to construct an M-alternating tour $(i_1, \ell_2) \cup P_2 \cup \cdots \cup (i_m, \ell_{m+1}) \cup P_{m+1}$. But Lemma 3.11 then tells us that all vertices i_j and ℓ_{j+1} for $j = \{1, \ldots, m\}$ are in the same component of H, which is a contradiction. □

3.2.1 Implementation

To construct a fine decomposition Ait-Aoudia et al. (1993) provided an efficient algorithm. See Algorithm 3.2.

Algorithm 3.2 An algorithm for the fine decomposition

```
Input: A well-constrained bipartite graph $G = (V, E)$ with $V = L \cup I$
Output: $q$ subgraphs $H_1, \ldots, H_q$ with $V(H_1) \cup \cdots \cup V(H_q) = V$
1 find a maximum matching $M$ of $G$
2 directed graph $G' \leftarrow (V, \emptyset)$
3 foreach edges $(\ell, i)$ in $G$ do
4     add arc $(\ell, i)$ to $G'$
5 foreach edges $(\ell, i)$ in $M$ do
6     add arc $(i, \ell)$ to $G'$
7 directed subgraphs $G'_1, \ldots, G'_q \leftarrow$ strongly connected components of $G'$
8 for $j$ is 1 to $q$ do
9     $H_j \leftarrow G\left[V(G'_j)\right]$
```

This algorithm follows directly from Lemma 3.11. Every strongly connected component of G' corresponds to either a single match (or edge) from M or an M-alternating tour.

Lines 2 to 6 and lines 8 and 9 run in $O(n + m)$ time where n is the number of vertices and m is the number of edges in G. Line 7 also runs in $O(n + m)$ by using *Tarjan's Algorithm*, see Tarjan (1972). The complexity of the whole algorithm is determined by the complexity of finding a maximum matching in line 1. This can be done in $O(m\sqrt{n})$ time using the bipartite matching algorithm of Hopcroft and Karp (1973).

Appendix B contains a recursive and a non-recursive version of Tarjan's Algorithm. A nice property of using Tarjan's Algorithm is that the order in which the strongly connected components are found is an order of resolution for the subsystems $H_1, \ldots, H_q$.

Notice that lines 1 to 6 of Algorithm 3.1 and Algorithm 3.2 are the same. A consequence is that both algorithms can be merged very efficiently. However, one must be aware of the fact that the input of the two algorithms differ. But when running Algorithm 3.2 with V_1 from Algorithm 3.1 as input one can use $E(V_1) \cap M$ as matching and $G'[V_1]$ as directed graph.

3.3 The analysis component

Now that we have more information about the properties of the Dulmage-Mendelsohn, let us focus again on the interactive solver.

The interactive solver gives the current system of equations and its associated bipartite graph to the *analysis component*. This component uses the Dulmage-Mendelsohn decomposition to divide the system into the three subsystems V_1, V_2 and V_3. V_1 is well-constrained, V_2 is over-constrained and V_3 is under-constrained.

If the analysis component finds a non-empty V_2, it stops and returns this system to the solver. At least one of the equations in V_2 is either redundant or contradicting and should be disabled by the solver. After that the system can be fed to the analysis component again.

After this check the analysis component is left with V_1 and V_3. It tries to find irreducible components in V_1 using the method described in Subsection 3.2.1. Then the solver determines which of the subsystems associated to the irreducible components can be solved without solving other subsystems first.

Finally the analysis component provides the solver with two lists: a list of solvable subsystems of V_1 and a list of free parameters in V_3 that can be guessed without creating a non-empty V_2 in the resulting system of equations. By definition, all parameters in V_3 have this property. The solver can either solve a subsystem using the partial solver of Section 4.2 or assign a (random) value to one of the free parameters. After this action the system can be analyzed by the analysis component again. Note that both actions will not introduce a non-empty V_2 component when the modified system of equations will be analyzed again.

4 Newton methods

To solve a system of quations as described in Sections 2.4 and 3.3, the interactive solver has to solve (well-constrained) (sub)-systems of nonlinear equations. In this chapter we shortly describe the Newton method.

The *Newton method* is the most important algorithmic concept for solving a system of nonlinear equations

$$h(x) = 0 \tag{4.1}$$

where $h : \mathbb{R}^n \to \mathbb{R}^n, h \in C^1(\mathbb{R}^n, \mathbb{R}^n)$ is a system of n continuously differentiable (nonlinear) functions of n variables.

The classical *Newton iteration* for computing a solution $\bar{x}$ of (4.1) is

$$x_{k+1} = x_k - J_h(x_k)^{-1} h(x_k) \tag{4.2}$$

where J_h is the $n \times n$ Jacobian matrix of h and $x_0 \in \mathbb{R}^n$ is a (random) starting vector. To find x_{k+1} we simply have to solve the linear system

$$J_h(x_k)(x_{k+1} - x_k) = -h(x_k) \tag{4.2$'$}$$

for $x_{k+1} - x_k$ and then calculating x_{k+1} from x_k.

It is well-known that the Newton iteration (4.2) is locally quadratically convergent to a solution $\bar{x}$ of $h(\bar{x}) = 0$ if the regularity condition

$$J_h(\bar{x}) \text{ is nonsingular} \tag{4.3}$$

is satisfied, i.e. under the assumption of (4.3) it holds that for any starting point x_0 sufficiently close to $\bar{x}$, the iterates x_k of (4.2) converge to $\bar{x}$ with a rate

$$\|x_{k+1} - \bar{x}\| \leq c \, \|x_k - \bar{x}\|^2 \tag{4.4}$$

where c is a constant (see e.g. Faigle et al. (2002) for a proof).

The Newton method has one major drawback: the computation of the Jacobian J_h is required. This is not possible in the SST framework because the equations are implemented as "black box" functions. Besides that, every Newton iteration of (4.2) (or (4.2′)) is at least of computational complexity $O(n^3)$.

It would be better to have a method with the following features:

(a) only the values of $h(x_k)$ are required.

(b) the computational complexity of each iteration is smaller than $O(n^3)$, a better computational complexity would be $O(n^2)$.

(c) the iterates x_k are super-linear convergent, i.e.

$$\frac{\|x_{k+1} - \bar{x}\|}{\|x_k - \bar{x}\|} \to 0 \text{ as } k \to \infty.$$

The so-called quasi-Newton method as described in the following section possesses these properties.

4.1 The quasi-Newton method

The *quasi-Newton method* is similar to the Newton method but it uses

$$x_{k+1} = x_k + \alpha_k d_k$$

as update formula, where $d_k = -B_k^{-1} h(x_k)$, B_k is an approximation of $J_h(x_k)$ and α_k is some step size.

To calculate B_{k+1} one uses a low rank update $B_{k+1} = B_k + E_k$ where E_k is of low rank (like $\text{rank}(E) \leq 2$). The iterates B_{k+1} satisfy

$$B_{k+1}(x_{k+1} - x_k) = h(x_{k+1}) - h(x_k) \tag{4.5}$$

which is called the *quasi-Newton condition*. This condition can also be written as $B_{k+1} s_k = y_k$ where

$$\begin{aligned} s_k &= x_{k+1} - x_k \\ y_k &= h(x_{k+1}) - h(x_k). \end{aligned}$$

To motivate the quasi-Newton condition (4.5) consider the Taylor expansion around x_{k+1}:

$$h(x_k) - h(x_{k+1}) = J_h(x_{k+1})(x_k - x_{k+1}) + o(\|x_k - x_{k+1}\|).$$

So in (4.5) we obviously assume that B_{k+1} is an approximation of $J_h(x_{k+1})$ which satisfies the linear approximation

$$h(x_k) - h(x_{k+1}) \approx J_h(x_{k+1})(x_k - x_{k+1}).$$

The conceptual quasi-Newton method is described in Algorithm 4.1.

Algorithm 4.1 A conceptual algorithm for the quasi-Newton method

```
Input: A function $h : \mathbb{R}^n \to \mathbb{R}^n, h \in C^1(\mathbb{R}^n, \mathbb{R}^n)$, a point $x_0 \in \mathbb{R}^n$ and a
       number of iterations $m$
Output: A vector $x_m \in \mathbb{R}^n$
1 $B_0 \leftarrow$ an approximation of $J_h$
2 for $k$ is $0$ to $m-1$ do
      // update $x_{k+1}$
3     $d_k \leftarrow -B_k^{-1} h(x_k)$
4     $x_{k+1} \leftarrow x_k + d_k$
      // update approximation of Jacobian
5     $B_{k+1} \leftarrow B_k + E_k$                     // or $B_{k+1}^{-1} \leftarrow B_k^{-1} + \tilde{E}_k$
```

To possibly enlarge the region of attraction instead of line 4 of Algorithm 4.1 we can perform a step with line-minimization

$$\begin{aligned} &\alpha_k \leftarrow \text{ a solution of } \min_{\alpha \in \mathbb{R}} \|h(x_k + \alpha d_k)\|^2 \\ &x_{k+1} \leftarrow x_k + \alpha_k d_k \end{aligned} \tag{4.6}$$

An update formula of particular interest is due to Broyden (1965). His famous update formula

$$B_{k+1} = B_k + \underbrace{\frac{(y_k - B_k s_k)\, s_k^T}{s_k^T s_k}}_{E_k} \tag{4.7}$$

or equivalently

$$B_{k+1}^{-1} = B_k^{-1} + \underbrace{\frac{\left(s_k - B_k^{-1} y_k\right) s_k^T B_k^{-1}}{s_k^T B_k^{-1} s_k}}_{\tilde{E}_k} \tag{4.7'}$$

is known as Broyden's "good" update formula. As the name suggests he also proposed an update formula known as Broyden's "bad" update formula:

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)\, y_k^T B_k}{y_k^T B_k s_k} \tag{4.8}$$

or equivalently

$$B_{k+1}^{-1} = B_k^{-1} + \frac{\left(s_k - B_k^{-1} y_k\right) y_k^T}{y_k^T y_k}. \tag{4.8'}$$

Broyden (2000) gives the following explanation for the names of the formulas: the formula is referred to as "good" due to its better numerical performance relative to another formula that I also presented in (1965), which has become to be known as the "bad Broyden" update. Dennis and Schnabel (1980) discuss these two updates and their relations to the DFP and BFGS updates.

4.1.1 Convergence results

The quasi-Newton method with Broyden's update (4.7), also called *Broyden's method,* leads to a superlinearly convergent method.

Theorem 4.1 (Convergence result)**.** *Let h be a C^1-function in a (open) neighborhood of $\bar{x}$ such that $h(\bar{x}) = 0$ and $J_h(\bar{x})$ is nonsingular. Then for Broyden's method and the method from Algorithm 4.1 without using* (4.6) *the following holds.*

There exist constants $\delta, \epsilon > 0$ such that for any x_0, B_0 satisfying

$$\|x_0 - \bar{x}\| < \delta \text{ and } \|B_0 - J_h(x_0)\| < \epsilon$$

the iterates x_k converge to $\bar{x}$ superlinearly.

Proof. See Broyden et al. (1973). □

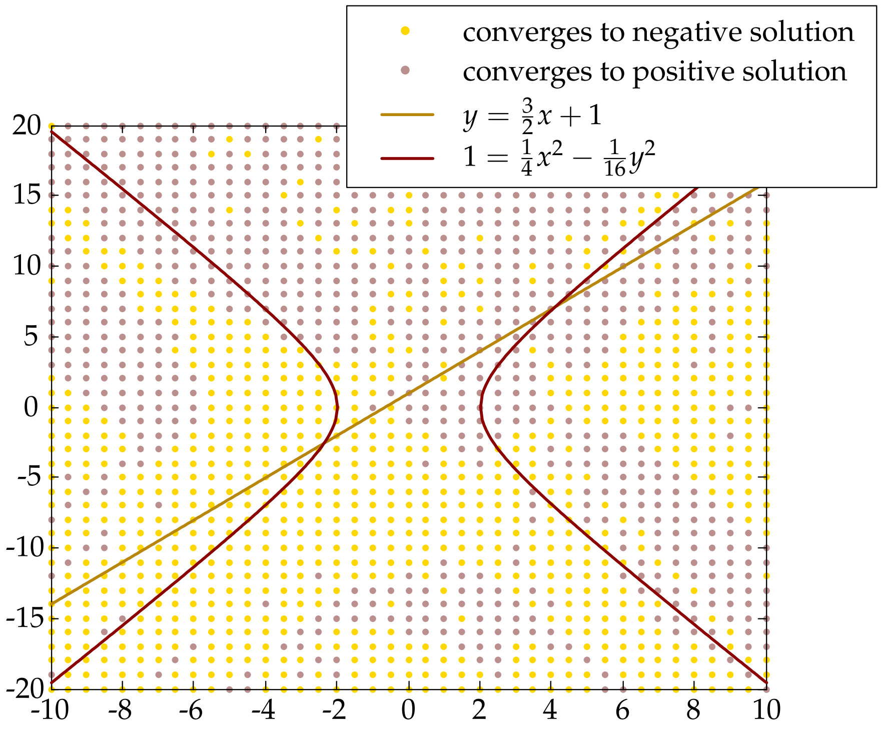


Figure 4.1: Behaviour of convergence of Broyden's method on System 4.10

Note that in the case where $n = 1$, the quasi-Newton condition

$$B_k s_{k-1} = y_{k-1} \text{ or } B_k^{-1} = \frac{s_{k-1}}{y_{k-1}} = \frac{x_k - x_{k-1}}{h(x_k) - h(x_{k-1})}$$

yields the so called *secant method*

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{h(x_k) - h(x_{k-1})} h(x_k). \tag{4.9}$$

It is also proved by Gay (1979) that when Broyden's method is applied to a linear system, it terminates in $2n$ steps. This is a usefull property for the solver. When a linear system has to be solved, there is no need to use a specialized method for linear systems.

Broyden's method is a "black box" method. As such, it is very hard to give a prediction of its convergence behaviour. Consider for example the system of equations

$$\left\{ \begin{array}{c} -\frac{3}{2}\,x_1 + x_2 = 1 \\ \frac{1}{4}\,x_1^2 - \frac{1}{16}\,x_2^2 = 1 \end{array} \right\} \tag{4.10}$$

which has $(-2.375, -2.563)$ and $(4.090, 7.134)$ as solutions. One could create a grid of starting points and determine for every starting point to which solution it converges. This has been done in Figure 4.1. There is no clear global convergence behaviour, except (according to Theorem 4.1) in a small neighbourhood of the solutions. An interesting sidenote is that Broyden's method converged for all starting points in Figure 4.1 in less then 29 iterations.

4.2 The partial solver

The *partial solver* is the component of the SST framework that used Broyden's method to solve the (irreducible) subsystems that are returned from the analysis compontent of Section 3.3. By definition these subsystems are well-constrained and thus consistent. That also means that with a "high probability" h is regular, i.e. all solutions $\bar{x}$ of $h(\bar{x}) = 0$ satisfy the regularity condition (4.3). So by Theorem 4.1, if solutions $\bar{x}$ of $h(\bar{x}) = 0$ exist, with high probability Broyden's method will converge to them when the starting point is close enough to the solution.

If a solution does not exist or the random starting point is too far from a solution, Broyden's method might not converge. If the iterates have not converged after a specified number of iterations, the partial solver will restart with another random starting point. After a specified number of restarts the partial solver will report to the solver that it was not able to solve the subsystem.

In this situation, probably the best choice for the interactive solver is to discard the current assignment of parameters, enable all rules again and start all over. Because the choices for assigned parameters and disabled rules have consequences for the type of encountered subsystems, it could be well possible that the interactive solver will be able to solve all encountered subsystems in the next attempt.

It may be noted that the interactive solver (as described in Sections 2.4, 3.3

and this section) has been implemented in the SST project. The implementation has been able to find feasible solutions for a set of test problems. However, as this implementation is outside the scope of the Final Project, it will not be further discussed within this thesis.

5 Bandwidth reduction

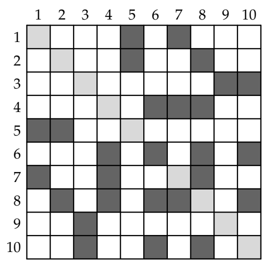
A major problem of manipulating large sparse matrices is that a lot of storage space and computing time are needed. One way of dealing with this is by permuting the rows and columns of the matrix such that it becomes a band matrix. A band matrix is a sparse matrix whose nonzero elements are within a certain distance from the main diagonal. When such permutations result in a band matrix of small bandwidth, considerable savings in both storage space and computing time are possible using banded algorithms, see for example Martin and Wilkinson (1967). A natural question that arises is what the smallest possible bandwidth for a given matrix is.

A lot of research has been put into reducing the bandwidth of symmetric matrices. However, unsymmetric matrices didn't get much attention. In the SST project unsymmetric matrices are more common, so this problem is worth investigating.

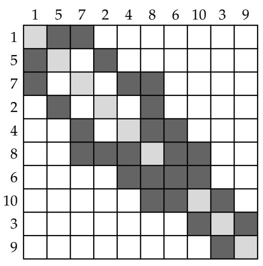
In Subsections 5.1 and 5.2, the symmetric and unsymmetric bandwidth minimization problem will be introduced respectively. Subsection 5.3 will give a short overview of the available literature of both problems. In Subsection 5.4 a proof will be presented showing that the unsymmetric bandwidth minimization problem is NP-complete. A popular class of algorithms concerning bandwidth reduction, the level set algorithms, will be treated in Subsection 5.5 and some more traditional heuristics are covered in Subsection 5.6. A short overview of the software written for this thesis can be found in Subsection 5.8 and finally, the computational results can be found in Subsection 5.7.

5.1 Symmetric case

Before the problem can be stated, a few definitions are needed.

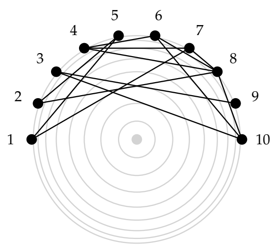


(a) Original with bandwidth 7

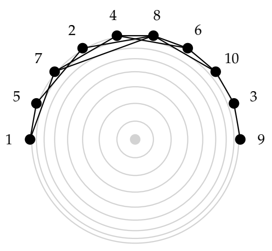


(b) Permuted with minimum bandwidth 3

Figure 5.1: Symmetric matrix



(a) Original with bandwidth 7



(b) Permuted with minimum bandwidth 3

Figure 5.2: Graph representation of Figure 5.1

Given a symmetric matrix $A = \left[a_{ij}\right] \in \mathbb{R}^{n \times n}$, the *semi-bandwidth* of matrix A is defined as $b_s = \max\left(|i-j| : a_{ij} \neq 0\right)$. In the literature different definitions for the bandwidth of a matrix are used. Usually the bandwidth is either b_s, $2b_s$ or $2b_s + 1$. In this thesis the *bandwidth* of a symmetric matrix is simply defined as $b = b_s$, the semi-bandwidth.

The *(symmetric) bandwidth minimization problem* (BMP) is defined as follows: given a symmetric matrix A a permutation of the rows and columns of A (where both permutations are the same to preserve symmetry) must be found such that the bandwidth, b, is minimized. In other words, all nonzero elements of A should be in a band that is as close as possible to the main diagonal.

In the context of graphs, given a graph $G = (V, E)$, where V is the vertex set with $|V| = n$ and E is the edge set, the bandwidth minimization problem is formulated as follows: find a bijective labeling $p : V \to \{1, \ldots, n\}$ that minimizes $\max\{|p(i) - p(j)| : (i,j) \in E\}$, or, equivalently, minimize the length of the longest edge when the vertices are ordered on a line with unit distance between consecutive vertices.

The matrix bandwidth minimization problem and the graph bandwidth minimization problem are interchangeable using A as the (vertex-vertex) adjacency matrix of G.

Figure 5.1 shows an instance of the symmetric bandwidth minimization problem. The nonzero elements of the matrix are shown as dark grey squares and the elements on the main diagonal are shown as light grey squares. Figure 5.2 shows the graph representation of the same problem. The points of the graph are ordered on half a circle instead of on a line to give a better visualization of the length of an edge.

5.2 Unsymmetric case

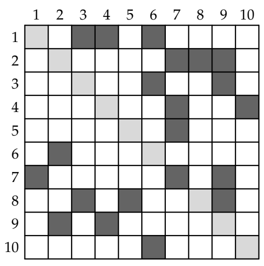
The main focus of this thesis will be a generalisation of the bandwidth minimization problem, namely the unsymmetric bandwidth minimization problem. In this problem, matrix A is allowed to be unsymmetric. While such a matrix is allowed to be non-square, the focus will be on square unsymmetric matrices.

Because of the loss of symmetry another definition for the bandwidth is needed. Given a matrix $A = [a_{\ell i}] \in \mathbb{R}^{m \times n}$ where $\ell \in \{1, \ldots, m\}$ and $i \in \{1, \ldots, n\}$ let us define the *lower bandwidth* b_ℓ as $\max\{\ell - i : a_{\ell i} \neq 0,\ \ell > i\}$ and the *upper bandwidth* b_u as $\max\{i - \ell : a_{\ell i} \neq 0,\ \ell < i\}$. Different definitions for the bandwidth are possible. Common choices are $\max(b_\ell, b_u)$, $b_\ell + b_u$, $b_\ell + b_u + 1$ and $b_\ell + b_u + \min(b_\ell, b_u)$. In this thesis the *bandwidth* b of an unsymmetric matrix is simply defined as $\max(b_\ell, b_u)$. Let us denote by $b(A)$ the bandwidth of matrix A. Note that for symmetric matrices, $b = \max(b_\ell, b_u) = b_s$.

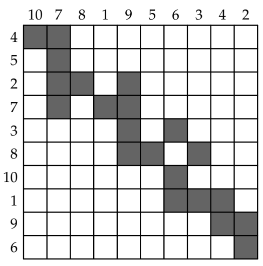
The *minimal bandwidth* of A is defined by

$$\beta(A) = \min_{\pi_L \in S_m,\, \pi_I \in S_n} b(\left[a_{\pi_L(\ell)\,\pi_I(i)}\right])$$

where S_n and S_m are the symmetric groups of permutations of respectively n and m objects. So $\beta(A)$ denotes the smallest bandwidth that can be achieved by permuting the rows and columns of A.

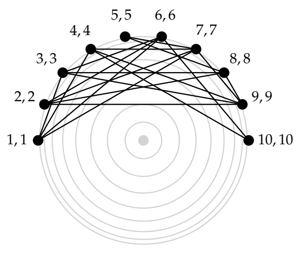


(a) Original with bandwidth 7

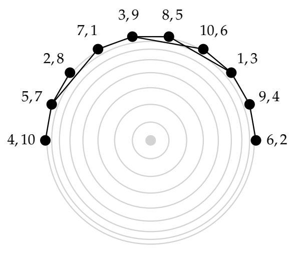


(b) Permuted with minimum bandwidth 2

Figure 5.3: Unsymmetric matrix



(a) Original with bandwidth 7



(b) Permuted with minimum bandwidth 2

Figure 5.4: Graph representation of Figure 5.3

The *unsymmetric bandwidth minimization problem* (UBMP) is to determine $\beta(A)$, given the matrix A.

In the context of graphs, consider a bipartite graph $G = (V, E)$, where $V = L \cup I$ is the vertex set and $E \subseteq L \times I$ is the edge set. L and I are the vertex classes of the bipartite graph with $|L| = m$ and $|I| = n$. The *bipartite graph bandwidth minimization problem* is to find a bijective labeling $p_L(v) : L \to \{1, \ldots, m\}$ and a bijective labeling $p_I(v) : I \to \{1, \ldots, n\}$ that minimizes $\max\{|p_L(\ell) - p_I(i)| : (\ell, i) \in E\}$, or, equivalently, minimize the length of the longest edge when the vertices are ordered with unit distance on two parallel lines, one for each vertex class, where the length $l(e)$ of an edge e is the Euclidean distance between its two incident vertices projected on either of the two lines.

The unsymmetric bandwidth minimization problem and the bipartite graph bandwidth minimization problem are interchangeable: each row ℓ of matrix A corresponds to a vertex $\ell \in L$ and each column i of matrix A corresponds to a vertex $i \in I$; $a_{\ell i}$ is nonzero iff $(\ell, i) \in E$.

Figure 5.3 shows an example of the unsymmetric bandwidth minimization problem. Figure 5.4 shows the graph representation of the same problem. Again, the vertices of the graph are ordered on a half circle instead of on a line to give a better visualization of the length of an edge. Furthermore, the vertex corresponding to row position j is placed at the same coordinates as the vertex corresponding to column position j. For example $3, 9$ means that both the vertices corresponding to row 3 and column 9 are located at that coordinate.

Note that the graph representation as used in Figure 5.4 is only usable with the bandwidth defined as $\max(b_\ell, b_u)$. When another bandwidth definition is used, one has to differentiate between edges where the column vertex is located to the left of the row vertex and edges where the column vertex is located to the right of the row vertex.

5.3 Literature

Most available literature covers the symmetric bandwidth minimization problem, which will be referred to as the BMP from now on. Since 1976 this problem is known to be NP-complete, due to Papadimitriou (1976). Unger (1998) even showed that, for any constant k, it is NP-complete to find a k-approximation of the BMP, i.e. finding a (polynomial) approximation algorithm that guarantees that the approximation will have a bandwidth smaller than $k\beta$ (k times the smallest possible bandwidth) would yield P=NP.

Nevertheless Corso and Manzini (1999) developed two algorithms to find an exact solution for the BMP. Because the running time of these algorithms can be very large for large problem instances, many efforts have been done to develop heuristic algorithms. These algorithms can be much faster, but don't guarantee optimal solutions. That is why they are usually referred to as *bandwidth reduction algorithms*.

A popular class of heuristic algorithms for the BMP is the class of *level set algorithms*. One of the most used algorithms in this catagory is the *reverse Cuthill-McKee algorithm (RCM)* which is simply the reverse order of the *Cuthill-McKee algorithm* by Cuthill and McKee (1969). The idea of the level sets in this algorithm led to the development of the *GPS algorithm* (by Lewis (1982)), the *Sloan algorithm* (by Sloan (1989)), the *JCL algorithm* (by Luo (1992)) and the more recent *WBRA algorithm* (by Esposito et al. (1998a)).

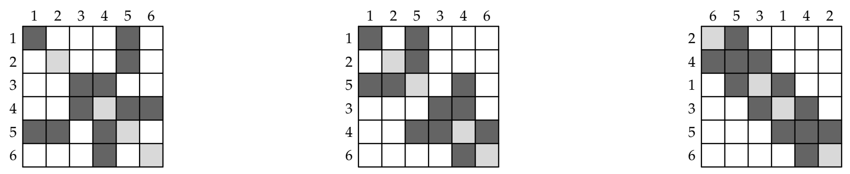
More recently, another class of algorithms for the BMP has been investigated: the class of metaheuristics. The first metaheuristic applied to the BMP was *simulated annealing*, due to Dueck and Jeffs (1995). A recent improvement of this approach has been proposed by Rodriguez-Tello et al. (2008). Esposito et al. (1998b) and Martí et al. (2001) used *tabu search* to reduce the bandwidth of a symmetric matrix. Piñana et al. (2004) presented the results of applying a greedy randomized adaptive search pocedure combined with a path relinking strategy (*GRASP-PR*). And finally, Pop and Matei (2011) developed a heuristic based on *genetic programming*.

As far as we know, little research has been done for the unsymmetric bandwidth minimization problem, which will be referred to as the UBMP from now on. Esposito et al. (1998b) describe an adjustment to Tabu Search and the WBRA algorithm (a heuristic for the BMP) to incorporate unsymmetric matrices. Reid and Scott (2006) developed methods to reduce a quantity called the *total bandwidth*, which is defined as $\min(b_\ell, b_u) + b_\ell + b_u$.

5.4 NP-completeness of the UBMP

The BMP is proved to be NP-complete, due to Papadimitriou (1976), by reducing 3-SAT to a generalized problem where a number of k edges are restricted by a bandwidth of $2b - 1$ instead of b. Another reduction is applied k times such that in the resulting problem all edges are restricted by a bandwidth of b'. In this section, the NP-completeness of the UBMP will be investigated and also proved by reducing 3-SAT to it. Some key concepts from the original proof will be used. However, a direct reduction from 3-SAT will be presented without using an intermediate problem.

Because the graph representations of the BMP and the UBMP (see Figures 5.2 and 5.4) look remarkably similar, it might be tempting to think that the UBMP generalizes the BMP and therefore is NP-complete. However, this is not the case. Consider for example Figure 5.5. Because of the fact that different permutations are possible for the rows and columns of the UBMP, there is extra freedom to improve the bandwidth.



(a) Original with bandwidth 4

(b) Symmetrically permuted with minimum bandwidth 2

(c) Unsymmetrically permuted with minimum bandwidth 1

Figure 5.5: Symmetric matrix

For the UBMP to be NP-complete, it has to be in NP. Recall that for a matrix $A = [a_{\ell i}]$, $\beta(A) = \min_{\pi_L \in S_m, \pi_I \in S_n} \max\left(|\pi_L(\ell) - \pi_I(i)| : a_{\ell i} \neq 0\right)$.

Definition 5.1. The *unsymmetric bandwidth minimization decision problem* (UBMDP) is the following: given a matrix $A \in \mathbb{R}^{m \times n}$ and an integer $b_d > 0$, is $\beta(A) \leq b_d$?

Lemma 5.2. *The UBMDP is in NP.*

Proof. Let the permutation of rows π_L and the permutation of columns π_I be the certificate of a "yes" instance. Now the "yes" instance can be verified by checking $|\pi_L(\ell) - \pi_I(i)| \leq b_d$ for all $\{\ell, i | a_{\ell i} \neq 0\}$. This can be done in $O(mn)$ time. □

Corollary 5.3. *The UBMP is an NP-optimization problem.*

Proof. By Lemma 5.2 the decision problem of the UBMP is in NP. □

To show that the UBMP is NP-complete, its NP-hardness must be proved as well. For this proof, a definition of 3-SAT is necessary. The *(exactly) 3-satisfiability problem* (3-SAT) is defined as follows: given n different Boolean variables $x_1, \ldots, x_n$ and r clauses $F_1, \ldots, F_r$ each containing exactly 3 different literals $f_1, f_2, f_3 \in \{x_1, \ldots, x_n, \bar{x_1}, \ldots, \bar{x_n}\}$, does an assignment of *TRUE* and *FALSE* to the variables $x_1, \ldots, x_n$ exist such that all clauses evaluate to *TRUE*?

An instance of 3-SAT with $n = 5$ and $r = 3$ could be for example

$$(x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge (\bar{x_1} \vee x_3 \vee \bar{x_4}) \wedge (x_2 \vee x_4 \vee \bar{x_5}) \,.$$

3-SAT is known to be NP-complete by Papadimitriou (1976).

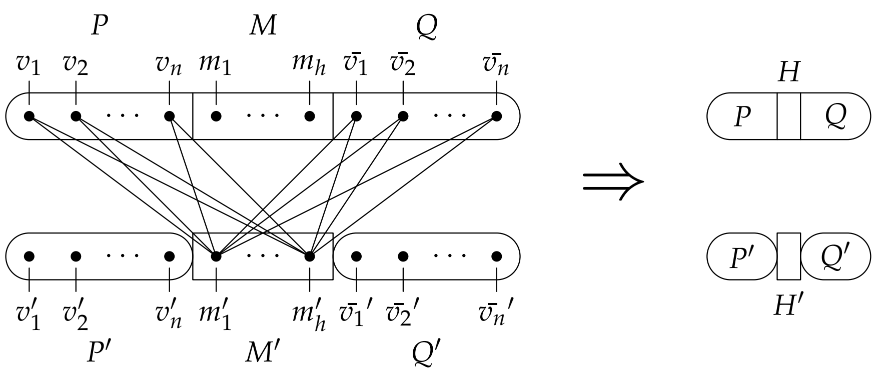

P
M
Q
v_1 v_2 v_n m_1 m_h $\bar{v_1}$ $\bar{v_2}$ $\bar{v_n}$
v'_1 v'_2 v'_n m'_1 m'_h $\bar{v_1}'$ $\bar{v_2}'$ $\bar{v_n}'$
P'
M'
Q'
$\Rightarrow$
H
P
Q
P'
Q'
H'

Figure 5.6: Bipartite graph (H, H')

Theorem 5.4. *The UBMP is NP-hard.*

Proof. By reducing 3-SAT to the decision problem UBMDP. Consider an instance S of 3-SAT with n Boolean variables $x_1, \ldots, x_n$ and r clauses $F_1, \ldots, F_r$. An instance U of the UBMDP with $b_d = n + 4$ will be created such that S is solvable iff U is a "yes" instance.

For a better understanding of the proof, the bipartite graph representation of the problem will be used. The most important building block of U will be the bipartite graph (H, H'), as shown in Figure 5.6. (H, H') can be constructed as follows: for each variable x_i in S, $i = 1, \ldots, n$, add 2 vertices v_i, $\bar{v}_i$ to H and v'_i, $\bar{v}_i{}'$ to H'. Also add the vertex sets $M = \{m_1, \ldots, m_h\}$ to H and $M' = \{m'_1, \ldots, m'_h\}$ to H' with $h = 2$ or $h = 3$. Now connect each v_i and $\bar{v}_i$ to all $m' \in M'$. The vertices v_i, $\bar{v}_i$ will be referred to as *literals*.

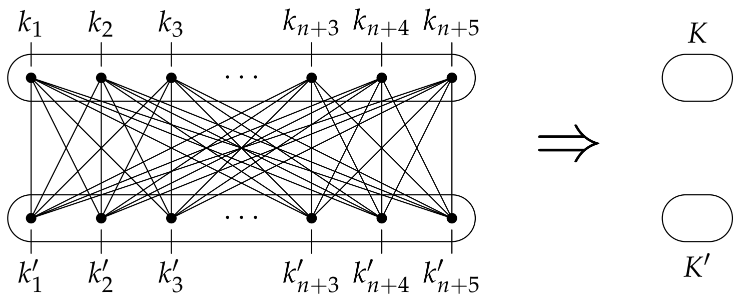
The rest of the construction of U as given below will force (H, H') to have certain properties in a valid solution of U:

(a) The vertices $M = \{m_1, \ldots, m_h\}$ will be next to each other in the middle of H and the vertices $M' = \{m'_1, \ldots, m'_h\}$ will be next to each other in the middle of H'. M will be positioned exactly above M'.

(b) Exactly n literals from $\{v_1, \ldots, v_n, \bar{v}_1, \ldots, \bar{v}_n\}$ are to the left of M, call this subset P, and exactly n literals are to the right of M, call this subset Q.

(c) Not both v_i and $\bar{v}_i$ are in P, i.e. v_i and $\bar{v}_i$ are at different sides of M.

(d) The assignment $x_i = TRUE$ iff $v_i \in Q$ is a valid solution of S.



k_1 k_2 k_3 k_{n+3} k_{n+4} k_{n+5}
k'_1 k'_2 k'_3 k'_{n+3} k'_{n+4} k'_{n+5}
K
K'

Figure 5.7: Bipartite graph (K, K')

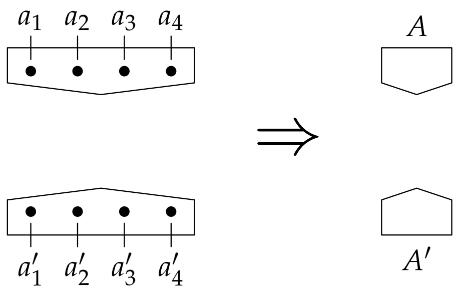


Figure 5.8: Bipartite graph (A, A')

Clearly the last property will be enough to prove the theorem. To enforce these properties, two more building blocks are needed. One of them is the bipartite graph (K, K') (see Figure 5.7). This is a complete bipartite graph with $K = \{k_1, \ldots, k_{n+5}\}$ and $K' = \{k'_1, \ldots, k'_{n+5}\}$. Because there is an edge between all vertices of K and all vertices of K', K will be exactly positioned above K' in a valid solution of U. Note that a bipartite graph consisting of one component can only be at one side of (K, K').

The last building block is the bipartite graph (A, A') (Figure 5.8). It consists of just $A = \{a_1, \ldots, a_4\}$ and $A' = \{a'_1, \ldots, a'_4\}$. The total construction of U will force A to be exactly positioned above A' in a valid solution of U. Also the vertices of A and A' will be positioned next to each other in a valid solution.

Now the bipartite graph U can be constructed. Take n copies of (H, H') with $h = 3$ to U and call them $(H_1, H'_1), \ldots, (H_n, H'_n)$ and add r copies of (H, H') with $h = 2$ and call them $(H_{n+1}, H'_{n+1}), \ldots, (H_{n+r}, H'_{n+r})$. Then add $n + r$ copies of (A, A') and 2 copies of (K, K'). Prearrange them in the

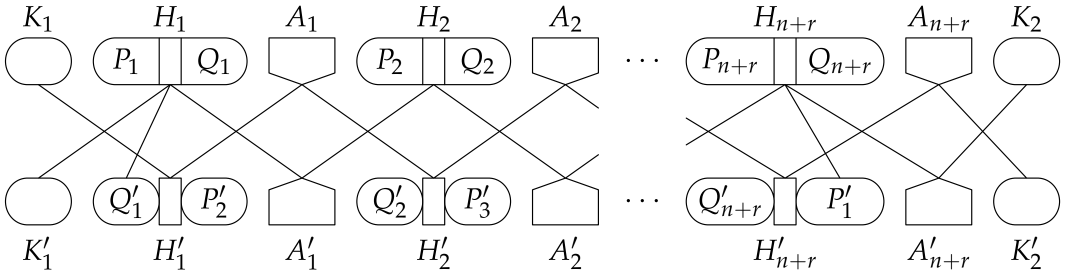


Figure 5.9: Prearrangement of U

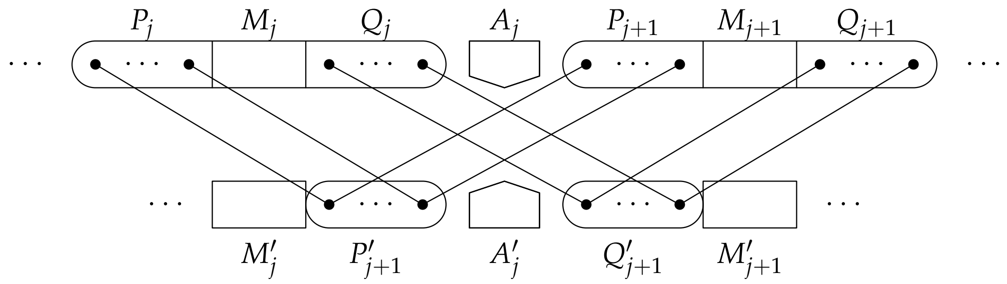


Figure 5.10: Edges added to U, where $j = 1, \ldots, n + r - 1$

order

$$(K_1, K'_1), (H_1, H'_1), (A_1, A'_1), \ldots, (H_{n+r}, H'_{n+r}), (A_{n+r}, A'_{n+r}), (K_2, K'_2).$$

Let us rearrange each bipartite graph (H_i, H'_i) such that (M_i, M'_i) is exactly in the middle of (H_i, H'_i). And within each vertex set H'_i replace each vertex set P'_i with Q'_i, Q'_j with P'_{j+1} and Q'_{n+r} with P'_1, where $i = 1, \ldots, n + r$ and $j = 1, \ldots, n + r - 1$, see Figure 5.9.

As Figure 5.9 suggests, as many edges as possible will be added between K_1 and M'_1, K'_1 and M_1, M_i and A'_i, M'_i and A_i, A_j and M'_{j+1}, A'_j and M_{j+1}, A_{n+r} and K'_2, A'_{n+r} and K_2, M_1 and Q'_1, and M_{n+r} and P'_1 under the condition that an added edge e will have a length $l(e) \leq n + 4$ given the current arrangement.

For each literal $v \in P_j \cup Q_j$, connect with it its corresponding literal $v' \in P'_{j+1} \cup Q'_{j+1}$ and connect v' with its corresponding literal $v'' \in P_{j+1} \cup Q_{j+1}$, as shown in Figure 5.10, where $j = 1, \ldots, n + r - 1$.

Let us consider instance U in its current form. Recall that in a valid solution of U there is no edge e with a length $l(e) > n + 4$. Note that every vertex not in (K_1, K_1') or (K_2, K_2') is part of a connected component adjacent to both (K_1, K_1') and (K_2, K_2'). Hence, in a valid solution of U, (K_1, K_1') and (K_2, K_2') must be arranged at the outer ends. Without loss of generality, we can say that (K_1, K_1') will be placed all the way to the left and (K_2, K_2') all the way to the right.

Because of the edges that were added in our construction, see Figure 5.9, and the fact that (K_1, K_1') and (K_2, K_2') will be arranged at the outer ends, the subgraphs

$$\left(M_1, M_1'\right), \left(A_1, A_1'\right), \left(M_2, M_2'\right), \ldots, \left(A_{n+r-1}, A_{n+r-1}'\right), \left(M_{n+r}, M_{n+r}'\right)$$

will stretch out leaving exactly n points of "free space" around them (this free space will be occupied by the vertices of P_i, Q_i, P_i' and Q_i'). This means that each subgraph (H_i, H_i') in U already has Property (a), with $i = 1, \ldots, n + r$. The edges that were added in Figure 5.6 restrict each subgraph (H_i, H_i') in U to have Property (b) and the property that vertices that were added in $P_i \cup Q_i$ will stay in $P_i \cup Q_i$ in a valid solution of U.

Furthermore, the edges added in Figure 5.10 force each partition of literals from $P_i \cup Q_i$ over P_i and Q_i to be the same for all subgraphs (H_i, H_i'). Note that the precise arrangement of literals within P_i and Q_i doesn't have to be the same for all subgraphs (H_i, H_i'). For $i = 1, \ldots, n-1$, due to the choice $h = 3$, a literal from H_i can move at most 1 position forward in H_{i+1}. And for $i = n, \ldots, n+r-1$, where $h = 2$, a literal from H_i can move at most 2 positions forward in H_{i+1}. This freedom will be necessary to enforce the last two properties. However, our construction will assure that the arrangement of literals within P_{j+1}' will be the same as within P_{j+1} and the arrangement within Q_{j+1}' the same as within Q_j, $j = 1, \ldots, n+r-1$.

To enforce Property (c), for each $i = 1, \ldots, n$, an edge is added between literal $v_i \in P_i \cup Q_i$ and $a_1' \in A_i'$, and between literal $\bar{v}_i \in P_i \cup Q_i$ and $a_1' \in A_i'$ (in every subgraph (H_i, H_i')). An illustration of these edges can be found in Figure 5.11. In a valid solution of U, the literal $\{v_i, \bar{v}_i\} \cap P_i$ will be positioned all the way to the right within P_i. When both v_i and $\bar{v}_i$ would be in P_i, one of the edges would have a length of at least $n + 5$ which is not allowed.

Finally, for Property (d), edges will be added such that each clause of S will evaluate to *TRUE* under the assignment as defined in Property (d). To this

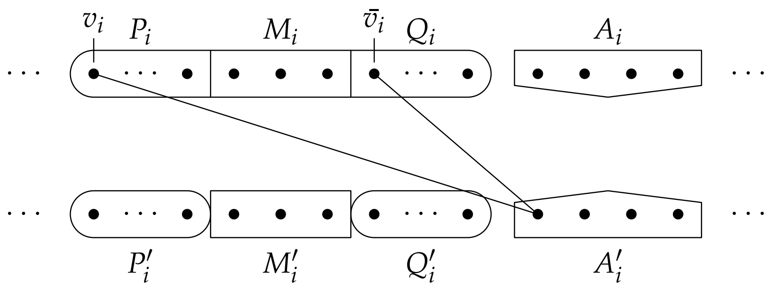


Figure 5.11: Edges added to U, where $i = 1, \ldots, n$

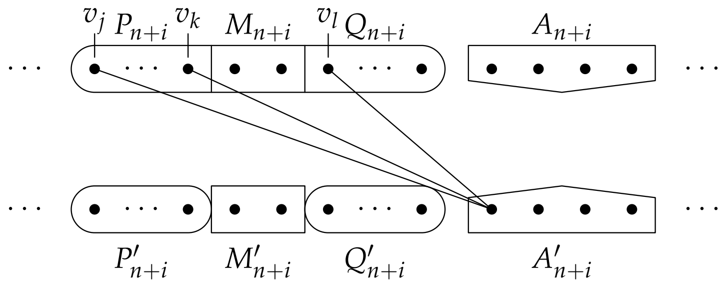


Figure 5.12: Edges added to U, where $i = 1, \ldots, r$

end, add an edge for each literal in clause F_i, $i = 1, \ldots, r$, as follows: if the literal is a Boolean variable x_j, add an edge between $v_j \in P_{n+i} \cup Q_{n+i}$ and $a'_1 \in A'_{n+i}$ and if the literal is a negation of a Boolean variable x_j, add an edge between $\bar{v}_j \in P_{n+i} \cup Q_{n+i}$ and $a'_1 \in A'_{n+i}$, where $j \in \{1, \ldots, n\}$. Note that for each subgraph (H_i, H'_i), exactly three edges are being added, as shown in Figure 5.12.

To see that this construction works, consider for example clause $F_j = x_1 \vee \bar{x}_2 \vee \bar{x}_3$ for a fixed j. The only assignment possible to evaluate this clause to *FALSE* is $x_1 = FALSE$, $x_2 = TRUE$ and $x_3 = TRUE$. This translates to a solution of U where v_1, $\bar{v}_2$ and $\bar{v}_3$ are in P_i, for all $i = 1, \ldots, n + r$. But because in (H_{n+j}, H'_{n+j}) all these vertices are connected to $a'_1 \in A'_{n+j}$, one of these edges must have a length of $n + 5$ concluding that this solution of U is not a valid solution.

The construction ensures that a valid solution of U implies a valid solution of S, by Property (d). Hence it remains to show that for a valid solution of S, a valid solution of U exists. Lets start with the same arrangement of vertices within U as at the end of the construction of U. Partition all literals of $P_i \cup Q_i$ in such a way over P_i and Q_i that $v_i \in Q_i$ iff $x_i = TRUE$ and not both v_i and $\bar{v}_i$ are in P_i, and move each $v \in P_i$ that is connected with $a_1 \in A'_i$ to the right within P_i, for $i = 1, \ldots, n+r$. Now it is relatively easy to find an arrangement where all edges e have a length $l(e) \leq n+4$ by arranging the literals of P_i in such a way that the corresponding literals that were moved to the right in P_{i+1} have a maximum distance of $2n+8$, where $i = 1, \ldots, n+r-1$. This is always possible because of the freedom that was decribed when adding the edges of Figure 5.10. Finally arrange the literals within P'_{j+1} the same way as within P_{j+1} and arrange the literals within Q'_{j+1} and Q_{j+1} the same way as within Q_1, where $j = 1, \ldots, n+r-1$. This results in a valid solution of U.

What remains to show is that the whole construction can be carried out in polynomial time. For an instance S of 3-SAT with n Boolean variables and r clauses an instance U of the UBMP is created with $(4n+12)\,r + 4n^2 + 18n + 20$ vertices and $(8n+31)\,r + 12n^2 + 67n + 70$ edges. It is clear that this can be done in polynomial time. □

5.4.1 Other bandwidth definitions

Until now only the UBMP with a bandwidth defined as $\max(b_\ell, b_u)$ was considered. This section investigates the consequences of using other definitions for the bandwidth on the NP-hardness of the UBMP, namely $b_\ell + b_u$, $b_\ell + b_u + 1$ and $b_\ell + b_u + \min(b_\ell, b_u)$. Clearly the UBMP remains in NP with these definitions of the bandwidth. In what follows, it will be shown that these definitions do not change the NP-hardness of the UBMP.

First, let us consider the case where the bandwidth is defined as $b = b_\ell + b_u$. An instance U_2 of the UBMP can be created in the same way as in Theorem 5.4. To differentiate between the lower bandwidth and the upper bandwidth, replace all edges with arcs pointing downwards. Without loss of generality we can say that the lower bandwidth b_ℓ is the maximum length of all arcs pointing left and the upper bandwidth b_u is the maximum length of all arcs pointing right. Naturally we search for an arrangement of the vertices where $b = b_\ell + b_u \leq 2\,(n+4) = 2n+8$. In a valid solution of U_2 where $b_\ell \leq n+4$ and $b_u \leq n+4$, we have a solution that would also be valid for U. This means that all properties described in Theorem 5.4 remain true and a valid solution of S can be easily constructed from a valid solution of U_2 and vice versa.

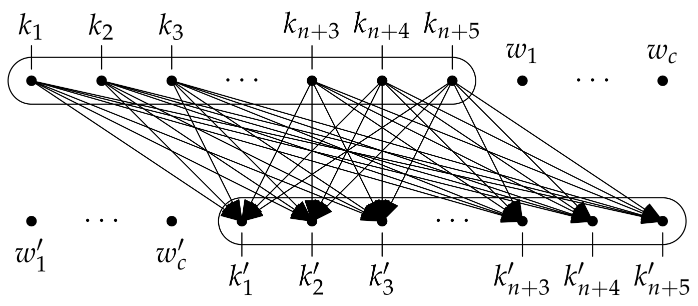


Figure 5.13: Shifted subgraph (K_1, K'_1), where $c \in \{1, \ldots, n+4\}$

However, we also might end up with a solution where $b_\ell \leq n + 4 - c$ and $b_u \leq n + 4 + c$, with $c \in \{-n-4, \ldots, n+4\} \setminus \{0\}$. The only arrangement for subgraph (K_1, K'_1) that is possible for such a solution is an arrangement where all the vertices of K_1 and K'_1 are next to each other and where the vertices of K'_1 are relatively positioned c positions to the right of the vertices of K_1, as in Figure 5.13. Note that under the current conditions, it is still impossible for an arc to start from a different side of (K_1, K'_1) than where it ends. And because every vertex in U_2 not in (K_1, K'_1) or (K_2, K'_2) is part of a connected component adjacent to both (K_1, K'_1) and (K_2, K'_2), it follows that in a valid solution of U_2, (K_1, K'_1) and (K_2, K'_2) must be arranged at the outer ends. This implies that $b_\ell \leq n + 4$ and $b_u \leq n + 4$ and we are done.

For the case where the bandwidth is defined as $b = b_\ell + b_u + 1$, note that this is exactly the same as the previous case when we search for an arrangement of the vertices where $b = b_\ell + b_u + 1 \leq 2\,(n+4) + 1 = 2n + 9$.

Finally, there is the case where the bandwidth is defined as $b = b_\ell + b_u + \min(b_\ell, b_u)$, also called the *total bandwidth* by Reid and Scott (2006). An instance U_3 of the UBMP can be created as in Theorem 5.4. Now we would search for an arrangement of the vertices where $b = b_\ell + b_u + \min(b_\ell, b_u) \leq 3\,(n+4) = 3n + 12$. The argument used before doesn't hold in this case because a solution where $b_\ell \leq 0$ and $b_u \leq 3n + 12$ would also be valid under this bandwidth definition and then it is possible for an arc to "jump over" (K_1, K'_1). Even more, the vertices of K_1 and K'_1 are not forced to be next to each other anymore. This means that the construction of Theorem 5.4 is not valid for this bandwidth definition directly.

Analyzing this bandwidth definition a bit more, one can see that this definition is actually a trade-off between minimizing $b_\ell + b_u$, which minimizes the (shifted) bandwidth resulting in savings in both storage space and computing time, and minimizing $\min(b_\ell, b_u)$, which forces the matrix to be as triangular as possible resulting in a more efficient Gaussian elimination. Because the latter minimization is a whole different problem, one could argue that another approach would be needed to prove its NP-hardness.

However, it is still possible to prove its NP-hardness by forcing the term $\min(b_\ell, b_u)$ to be zero. To achieve this, add $n+4$ vertices $w_1, \ldots, w_{n+4}$ and $n+4$ vertices $w'_1, \ldots, w'_{n+4}$ to U_3 and search for an arrangement of the vertices where $b = b_\ell + b_u + \min(b_\ell, b_u) \leq 2\,(n+4) = 2n+8$. Because of the subgraph (K_1, K'_1) we know that $b_\ell + b_u = 2n+8$ so $\min(b_\ell, b_u)$ must be zero. This implies that either b_ℓ or b_u is equal to $2n+8$. Now it is possible to show that the problem of U_3 is equivalent to the problem of U_2. Given a valid solution of U_2, add the vertices $w_1, \ldots, w_{n+4}$ and place them all the way to the left and add the vertices $w'_1, \ldots, w'_{n+4}$ and place them all the way to the right to get a valid solution of U_3. Given a valid solution of U_3, remove all vertices $w_1, \ldots, w_{n+4}, w'_1, \ldots, w'_{n+4}$ to get a valid solution of U_2.

5.4.2 Small errors in original proof

While investigating the proof of the NP-completeness of the BMP by Papadimitriou (1976), a few small errors were found that made it more difficult to comprehend and validate the construction. The most important will be listed below. All important errors are on the 6th page of the article (page 268 of the journal).

In the third paragraph, $a_{n+r}+1$ should be replaced with a_{n+r} and $p-n-m_1-1$ should be replaced with $p-n-m_1$. In the fourth paragraph, $m_i = 4$ should be $m_i = 3$ for $1 \leq i \leq n$ and $m_i = 3$ should be $m_i = 2$ for $n+1 \leq i \leq n+r$. In the fifth paragraph, *nodes n* should be replaced with *nodes N*. And finally, in the sixth paragraph, $\{(v_i, v_j) : |i-j| < p\}$ should be replaced with $\{(v_i, v_j) : |i-j| \leq p\}$ and $p-1$ should be replaced with p.

5.5 Level set algorithms

Now that the NP-completeness of the UBMP has been proved, some heuristics to reduce the bandwidth of an instance of the UBMP will be investigated. Since most available literature covers the BMP, first some of the heuristics that are available for the BMP are considered.

A popular class of algorithms for the BMP is the class of *level set algorithms*. They operate on the graph representation of the BMP. For clarity, let us restate the symmetric graph bandwidth minimization problem: given a graph $G = (V, E)$, a bijective labeling $p : V \to \{1, \ldots, n\}$ should be found that minimizes $\max\{|p(i) - p(j)| : (i, j) \in E\}$.

The main idea of the level set algorithms is to partition the vertices V over level sets $V_1, \ldots, V_m$ such that the cardinality of the level set with the largest cardinality is as small as possible and adjacent vertices are in the same or in consecutive level sets. Finally a labeling will be constructed by labeling all vertices within a level set in order of increasing degree and giving a larger label to vertices in a higher level set.

5.5.1 Cuthill-McKee algorithm

The first implementation of a level set algorithm was due to Cuthill and McKee (1969). The Cuthill-McKee algorithm is basically a breadth first search (BFS) where the neighbors of a vertex are visited in order of increasing degree. The order of visits determines a labeling that results in a relatively small bandwidth.

Algorithm 5.1 The Cuthill-McKee algorithm for the BMP

Input: A graph $G = (V, E)$ and a starting vertex $v \in V$
Output: A bijective labeling $p : V \to \{1, \ldots, n\}$

```
level sets $\{V_1, \ldots, V_m\} \leftarrow$ BFS$(G, v)$        // Algorithm 5.2
$i \leftarrow 1$
for $j \leftarrow 1$ to $m$ do
    foreach vertex $v$ in $V_j$ do                    // in order of occurrence
        $p(v) \leftarrow i$
        $i \leftarrow i + 1$
```

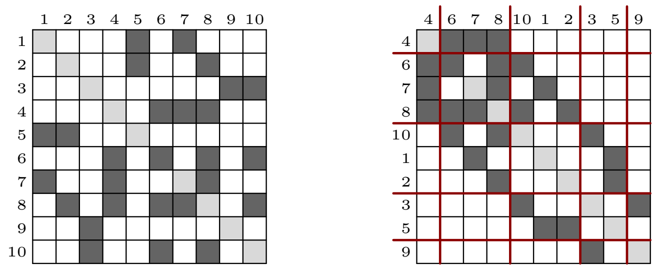
Algorithm 5.2 BFS (breadth first search) algorithm

Input: A graph $G = (V, E)$ and a starting vertex $v \in V$
Output: Level sets $\{V_1, \ldots, V_m\}$

```
$i \leftarrow 1$
set $v$ as visited
level set $V_i \leftarrow v$
add $v$ to queue $Q$
while $Q$ is not empty do
    while $Q$ is not empty do
        $v \leftarrow$ first element of $Q$
        remove first element of $Q$
        if $v \in V_{i+1}$ then
            $i \leftarrow i + 1$
        foreach neighbor $v'$ of $v$ do    // in order of increasing degree
            if $v'$ has not yet been visited then
                set $v'$ as visited
                add $v'$ to $V_{i+1}$
                add $v'$ to $Q$
    if $V$ contains unvisited vertices then
        $v' \leftarrow$ any unvisited vertex of $V$
        set $v'$ as visited
        $V_{i+1} \leftarrow v'$
        add $v$ to $Q$
```

Because the algorithm visits all vertices and edges at most twice, it runs in $O(|V| + |E|)$ time. Applying the Cuthill-McKee algorithm to a symmetric matrix returns an labeling (ordering) as in Figure 5.14. The vertices are ordered by increasing label. The extended (red) lines in the figure separate the vertices of different level sets. In this case, the starting vertex is the vertex corresponding to row/column 4. The choice of the starting vertex is important for the quality of the resulting ordering. Note that a starting vertex may not result in a unique labeling because different neighbors of a vertex may have the same degree.

The Cuthill-McKee algorithm works because the matrix is divided in blocks defined by the level sets. Because every edge is either between vertices within the same level set or between vertices in two consecutive level sets, the only nonzero elements of the matrix will be in the diagonal blocks and



(a) Original

(b) Permuted using the Cuthill-McKee algorithm

Figure 5.14: Applying Cuthill-McKee algorithm to a symmetric matrix

in the blocks next to the diagonal blocks. All other blocks will not contain nonzero elements. As a consequence, it is better for the bandwidth to have small level sets returned by the BFS.

A natural question is how to apply the Cuthill-McKee algorithm to the UBMP. The similarity of the graph representations of the BMP and the UBMP (see Figures 5.2 and 5.4) suggests the following method: given a graph $G = (V, E)$ with $V = L \cup I$ corresponding to an instance of the UBMP, the instance will be made "symmetric". First add a minimum number of vertices to V such that $|L| = |I|$. Let $L = \{\ell_1, \ldots, \ell_n\}$ and $I = \{i_1, \ldots, i_n\}$. Now construct a graph $G' = (V', E')$ such that $|V'| = |L|$. Let $V' = \{v'_1, \ldots, v'_n\}$. Add an edge $\left(v'_j, v'_k\right)$ to E' if and only if $\left(\ell_j, i_k\right) \in E$ or $\left(\ell_k, i_j\right) \in E$ for all $j, k \in \{1, \ldots, n\}$. The Cuthill-McKee algorithm can now be applied to graph G' and the resulting labeling can be used for both the row vertices L and column vertices I.

In term of matrices, consider a matrix $A = [a_{\ell i}]$ where $A \in \mathbb{R}^{m \times n}$, $\ell \in \{1, \ldots, m\}$ and $i \in \{1, \ldots, n\}$. Now add a minimum number of rows or columns to obtain a square matrix $\tilde{A}$. Then change a minimum number of "zero elements" of $\tilde{A}$ into "nonzero elements" such that the structure of zero and nonzero elements will be symmetric, for example $A' = \tilde{A} + \tilde{A}^T$. Finally the Cuthill-McKee algorithm can be performed on matrix A' and the resulting ordering can be used for both the rows and columns of A in an obvious way.

In fact, this exact method has been used by both Esposito et al. (1998b) and Reid and Scott (2006) to apply respectively the RCM and WBRA algorithms to unsymmetric matrices. However, this method has two main

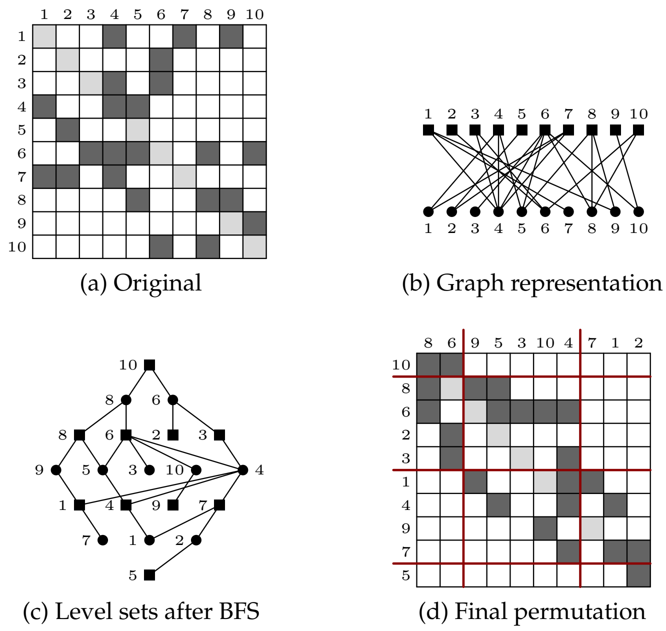

(a) Original (b) Graph representation (c) Level sets after BFS (d) Final permutation

Figure 5.15: Applying Cuthill-McKee algorithm to an unsymmetric matrix

drawbacks. Firstly, because of fill-in more elements of the matrix are considered as nonzero elements than necessary. This is especially true for (structurally) highly unsymmetric matrices. Secondly, this method does not exploit the extra freedom of using two different permutations for the rows and columns, as shown in Figure 5.5. Note that the first drawback can be partially overcome by prearranging the rows and columns of the matrix such that it will be as structurally symmetric as possible.

Another option is to apply the algorithm to the (symmetric) matrix instance $A' = \tilde{A}\tilde{A}^T$. This method has also been proposed by Esposito et al. (1998b) and Reid and Scott (2006), but suffers from the same drawbacks as the previous method.

A more direct method is preferable. This thesis will concentrate on algorithms focused on the bipartite graph representation of the UBMP. Actually, Reid and Scott (2006) also proposed this method and applying the Cuthill-McKee algorithm to the bipartite graph (see Algorithm 5.3) gives rather good results.

Because the input of the algorithm is a bipartite graph, for each level set V_j computed by Algorithm 5.3 it holds that either $V_j \subseteq L$ or $V_j \subseteq I$ with

Algorithm 5.3 The Cuthill-McKee algorithm for the UBMP

```
Input: A bipartite graph $G = (V, E)$ with $V = L \cup I$ and
       a starting vertex $v \in V$
Output: The bijective labelings $p_L(v) : L \to \{1, \ldots, m\}$ and
        $p_I(v) : I \to \{1, \ldots, n\}$
1 level sets $\{V_1, \ldots, V_q\} \leftarrow$ BFS$(G, v)$              // Algorithm 5.2
2 labelings $\{p_L, p_I\} \leftarrow$ Numbering$(\{V_1, \ldots, V_q\})$   // Algorithm 5.4
```

Algorithm 5.4 Numbering algorithm for the UBMP

```
Input: Level sets $\{V_1, \ldots, V_q\}$
Output: The bijective labelings $p_L(v) : L \to \{1, \ldots, m\}$ and
        $p_I(v) : I \to \{1, \ldots, n\}$
1  $i_L \leftarrow 1$
2  $i_I \leftarrow 1$
3  for $j \leftarrow 1$ to $q$ do
4      foreach vertex $v$ in $V_j$ do                // in order of occurence
5          if $v \in L$ then
6              $p_L(v) \leftarrow i_L$
7              $i_L \leftarrow i_L + 1$
8          else
9              $p_I(v) \leftarrow i_I$
10             $i_I \leftarrow i_I + 1$
```

$j = 1, \ldots, q$. Figure 5.15 clearly illustrates this with an example. Figure 5.15b shows the bipartite graph representation of the matrix in Figure 5.15a. The vertices corresponding to the rows and columns are represented as squares and circles respectively. The level sets obtained from the breadth first search starting from row vertex 10 can be found in Figure 5.15c. Each horizontal line of vertices represents a different level set starting from $V_1 = \{\text{row } 10\}$ above. Finally, Figure 5.15d shows the resulting permutation of the matrix.

Numerical results of applying this algorithm to a set of test problems can be found in Section 5.7.

5.5.2 COBRA algorithm

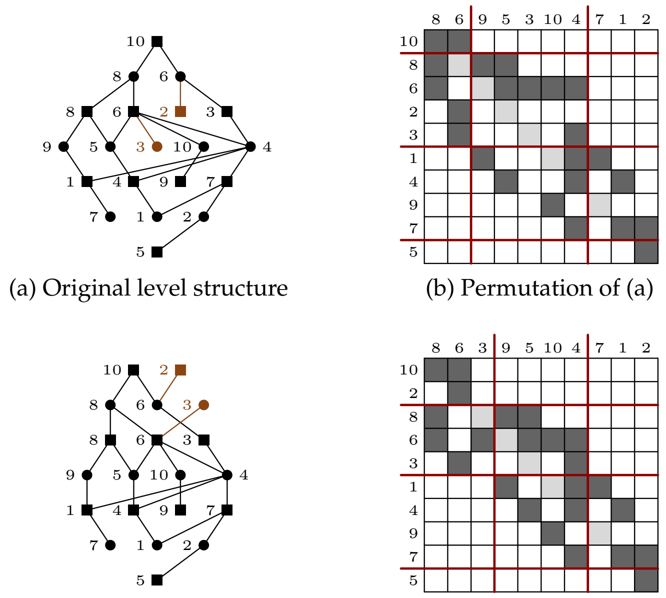
Applying the Cuthill-McKee algorithm to the bipartite graph representation of the UBMP has several disadvantages. For example the resulting ordering is very sensitive to the chosen starting vertex and the level sets can still be improved. We developed an algorithm to overcome these disadvantages, the *Chain Ordering Bandwidth Reduction Algorithm* (*COBRA*).

The main improvement is a heuristic that is applied to the level sets obtained from the BFS, the so-called *PushUp* heuristic. This heuristic has been previously described by Esposito et al. (1998a) for the BMP. Here we present a modified version for the UBMP. Let us define a *level structure* as a set of level sets. For example, the output of the BFS is a level structure. Now let the *width* of a level structure be the cardinality of its largest level set. As we may hope, a level structure of smaller width usually results in orderings of smaller bandwidth.

Let q be the number of level sets in a level structure. The PushUp heuristic for the UBMP tries to move a vertex $v \in V_{j+2}$ two level sets up in the level structure to level set V_j, where $j \in \{1, \ldots, q-2\}$, under two conditions. Firstly, the maximum cardinality of the two level sets must decrease, i.e. $|V_{j+2}| - |V_j| > 1$. Secondly, the neighbors of v must remain in adjacent level sets, i.e. $v' \notin V_{j+3}$ for all $(v, v') \in E$. The heuristic will keep moving vertices until no vertex can be moved anymore under the mentioned conditions.

Let d be the maximum degree of the vertices in V. Then the PushUp heuristic runs in $O(qd\,|V|)$ time. An example of using the PushUp heuristic can be found in Figure 5.16. Both row vertex 2 and column vertex 3 are pushed up. The width of the level structure decreases by 1 and in this case, the bandwidth of the matrix after permuting decreases from 4 to 3.

To overcome the problem of the sensitivity of the starting vertex on the resulting bandwidth, an inelaborate method has been used. Numerical experimentation revealed that applying the Cuthill-McKee algorithm with PushUp heuristic multiple times works rather good when the starting vertices are chosen randomly each time. The smallest obtained bandwidth rapidly approaches the smallest bandwidth possible using the Cuthill-McKee algorithm with PushUp heuristic. Therefore, for the COBRA algorithm just applies the Cuthill-McKee algorithm with PushUp heuristic a fixed num-



(a) Original level structure (b) Permutation of (a)

(c) After PushUp heuristic (d) Permutation of (c)

Figure 5.16: Applying the PushUp heuristic

ber of times t and returns the labeling with the best bandwidth, as shown in Algorithm 5.5. The whole algorithm runs in $O(qdt\,|V| + t\,|E|)$ time.

Numerical results of applying the COBRA algorithm to a set of test problems can be found in Section 5.7.

5.6 Metaheuristics

Besides level set algorithms, *metaheuristics* also became popular for the BMP recently. In this section two metaheuristics will be applied to instances of the UBMP: hill climbing in Subsection 5.6.1, and simulated annealing in Subsection 5.6.2.

The class of metaheuristics is a class of general purpose local search algorithms particularly useful for combinatorial optimization problems with a large set of feasible solutions. They work by iteratively trying to change a *feasible solution* $s \in S$ of a problem to minimize an *evaluation function* $f(s) : S \to \mathbb{R}$, where S is the set of all feasible solutions. To find an adjustment a *neighbor function* $n(s) : S \to S^r$ is used that generates r different feasible solutions, the *neighbors*, that are usually obtained by changing s

Algorithm 5.5 The COBRA algorithm for the UBMP

```
Input: A bipartite graph $G = (V, E)$ with $V = L \cup I$ and
       a number of trials $t$
Output: The bijective labelings $p_L(v) : L \to \{1, \dots, m\}$ and
        $p_I(v) : I \to \{1, \dots, n\}$
integer $b \leftarrow m + n$   // The variable holding the smallest bandwidth
labelings $\{p_L, p_I\}$       // The labeling with the smallest bandwidth
for $j \leftarrow 1$ to $t$ do
    vertex $v \leftarrow$ Random$(V)$                    // $v$ is a random vertex
    level sets $\{V_1, \dots, V_q\} \leftarrow$ BFS$(G, v)$         // Algorithm 5.2
    $\{V_1, \dots, V_q\} \leftarrow$ PushUp$(\{V_1, \dots, V_q\})$   // The PushUp heuristic
    labelings $\{p'_L, p'_I\} \leftarrow$ Numbering$(\{V_1, \dots, V_q\})$   // Algorithm 5.4
    if Bandwidth$(\{p'_L, p'_I\}) < b$ then
        $b \leftarrow$ Bandwidth$(\{p'_L, p'_I\})$
        $\{p_L, p_I\} \leftarrow \{p'_L, p'_I\}$
```

in some way. In this thesis, the neighbor function will always return one neighbor, so $r = 1$.

Given the set of neighbors, the heuristic will pick one of them and accepts the neighbor given some criteria. If the neighbor is accepted, it will be used as feasible solution s for the next iteration. The final goal is to minimize $f(s)$.

Both the neighbor function and the acceptance criteria have a big influence on the effectiveness of the metaheuristic. Numerical results of the mentioned metaheuristics can be found in Section 5.7.

The initial feasible solution that is used in this section is the matrix $s = A = [a_{\ell i}] \in \mathbb{R}^{m \times n}$ where $\ell \in \{1, \dots, m\}$ and $i \in \{1, \dots, n\}$. The set of all feasible solutions S will be all matrices that can be obtained by permuting the rows and columns of A. The neighbor function $n(s)$ works by either swapping two rows or two columns of s. These rows and columns are chosen in a special way. Recall that for the UBMP we consider $\max(b_\ell, b_u)$ as the bandwidth definition, i.e. let $b(s) = \max(|\ell - i| : a_{\ell i} \neq 0)$ be the current bandwidth of a solution s. Call an element $a_{\ell i}$ of s *critical* when $|\ell - i| = b(s)$. Let $C(s) = \{a_{\ell i} : |\ell - i| = b(s)\}$ be the set of all critical elements in s. Now the rows or columns to swap are chosen in such a way that it always swaps at least one critical element $c = a_{\ell i}$ and for this critical element it holds that $|\ell - i| \leq b(s)$ where ℓ and i are the indices of c after the swap. Note that only the critical element c will not increase the bandwidth, all other swapped nonzero elements may increase the bandwidth.

And finally, the evaluation function used in this section is

$$f(s) = b(s) + \frac{|C(s)|}{m+n}.$$

The motivation for this evaluation function is that the heuristic tries to minimize the bandwidth $b(s)$, but also tries to minimize the number of critical elements $|C(s)|$ when the bandwidth stays the same. Note that $|C(s)|$ is always smaller than $m+n$.

5.6.1 Hill climbing

Probably the simplest metaheuristic is called *hill climbing*. It only accepts a neighbor s' when s' is better than the current solution s, i.e. $f(s') < f(s)$. While the name suggests that it should be used for maximization, hill climbing can also be used for minimization problems by searching for a maximal $-f(s)$.

Hill climbing is good at finding a *local optimum*. A local optimum is a solution for which no improving neighbor exists. However, a local optimum may still be far from the *global optimum* s_{opt}, the best solution possible, i.e. $f(s_{\text{opt}}) = \min(f(s) : s \in S)$.

5.6.2 Simulated annealing

Because hill climbing can easily get stuck in a local optimum, several metaheuristics have been developed that are able to escape from local optima. One of these metaheuristics is *simulated annealing*. Simulated annealing was independently described by Kirkpatrick et al. (1983) and by Cerný (1985).

The name of the algorithm comes from the process of annealing in metallurgy and material science. Annealing is used to change several properties of a material by heating a material and then slowly let it cool down. The goal of annealing is to decrease the internal energy of the material. It works because the atoms of the material get diffused because of the heat. Then the slow cooling of the material lets the atoms find locations such that the internal energy of the material becomes smaller.

A similar process is used in simulated annealing to decrease the evaluation function $f(s)$, see Algorithm 5.6. In the algorithm, the function Random() returns a random value $r \in [0, 1]$.

Algorithm 5.6 Simulated annealing

Input: An initial solution s, a number of steps t, the temperatures T_{start} and T_{stop} and a cooling schedule $T(T_{\text{start}}, T_{\text{stop}}, i, t)$
Output: A final solution s_{best}

```
1  solution $s_{\text{best}} \leftarrow s$                                    // The best solution
2  for $i \leftarrow 1$ to $t$ do
3      neighbor $s' \leftarrow n(s)$
4      if $f(s') < f(s)$ then
5          $s \leftarrow s'$
6      else
7          temperature $T \leftarrow T(T_{\text{start}}, T_{\text{stop}}, i, t)$
8          if $T \neq 0$ and Random() $< e^{(f(s)-f(s'))/T}$ then
9              $s \leftarrow s'$
10     if $f(s) < f(s_{best})$ then
11         $s_{\text{best}} \leftarrow s$
```

Different cooling schedules are possible. For this thesis three different cooling schedules are implemented: T_{linear}, T_{slow} and T_{fast}. T_{linear} is just a linear cooling schedule, T_{slow} starts slowly in the beginning and cools down faster and faster and T_{fast} cools fast in the beginning and slows down at the end. To achieve these properties, we defined the cooling schedules as

$$\begin{aligned}
T_{\text{linear}}(T_{\text{start}}, T_{\text{stop}}, i, t) &= T_{\text{start}} + \frac{i}{t}\left(T_{\text{stop}} - T_{\text{start}}\right) \\
T_{\text{slow}}(T_{\text{start}}, T_{\text{stop}}, i, t) &= T_{\text{start}} + \left(1 - e^{-5}\right)^{-1}\left(e^{-5i/t} - e^{-5}\right)\left(T_{\text{stop}} - T_{\text{start}}\right) \\
T_{\text{fast}}(T_{\text{start}}, T_{\text{stop}}, i, t) &= T_{\text{start}} + \left(1 - e^{-5}\right)^{-1}\left(e^{-5+5i/t} - e^{-5}\right)\left(T_{\text{stop}} - T_{\text{start}}\right)
\end{aligned}$$

as shown in Figure 5.17.

The probability that a neighbor that is worse than the current solution will be accepted is $e^{(f(s)-f(s'))/T}$. Usually T_{stop} is zero. As $f(s) - f(s')$ is usually minus one in our application, we need to find a T_{start} such that $e^{-1/T_{\text{start}}}$ is

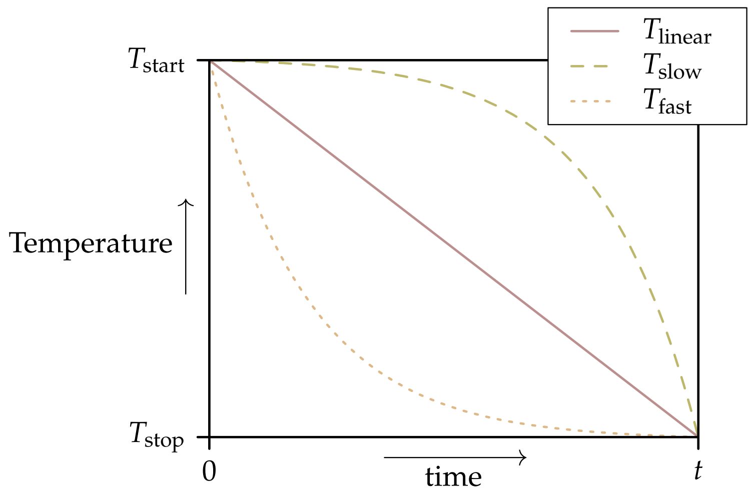


Figure 5.17: Different cooling schedules

big enough to accept a large fraction of worse neighbors in the beginning. Numerical experimentation revealed that $T_{\text{start}} = 10$ is a good value and $e^{-1/10} \approx 0.90484$.

5.7 Computational results

In this section some numerical results of running the algorithms from Sections 5.5 and 5.6 on a set of test problems will be presented. Each test problem consists of a matrix that is either computer generated (they start with the letter m) or emerged from a real engineering or industrial application. The latter are available through the University of Florida Sparse Matrix Collection, see Davis and Hu, and were also used by Reid and Scott (2006).

Table 5.1 shows some of the main characteristics of the matrices. The density is the number of nonzero elements divided by the total number of elements and the symmetry is the number of off-diagonal nonzero elements $a_{\ell i}$ where $a_{i\ell}$ is also nonzero divided by the total number of off-diagonal nonzero elements. Matrix `m200` is a randomly generated 200×200 matrix and `m200s` is the symmetric version of `m200` by mirroring its upper triangle into its lower triangle. Matrix `m500b80` has been created by randomly filling approximately half of the entries of a 500×500 matrix within a distance of 80 from the main diagonal. After that, the matrix has been randomly shuffled. Matrix `m1000` has also been randomly generated, and `m1000s` is its

Name	Size	Density	Symmetry	Bandwidth
`m200`	200×200	0.0133	0.0151	191
`m200s`	200×200	0.0126	1.0000	191
`m500b80`	500×500	0.1477	0.1521	499
`m1000`	1000×1000	0.0099	0.0085	986
`m1000s`	1000×1000	0.0100	1.0000	984
`bayer03`	6747×6747	0.0012	0.0031	6746
`circuit_3`	12127×12127	0.0003	0.7701	12077
`extr1`	2837×2837	0.0014	0.0042	2836
`hydr1`	5308×5308	0.0008	0.0041	5307
`impcol_d`	425×425	0.0074	0.0567	406
`jan99jac020sc`	6774×6774	0.0008	0.0038	6528
`poli_large`	15575×15575	0.0001	0.0035	15574
`radfr1`	1048×1048	0.0121	0.0537	948
`rdist1`	4134×4134	0.0055	0.0588	3934
`sinc15`	11532×11532	0.0043	0.0138	10937
`Zhao2`	33861×33861	0.0001	0.9225	32907

Table 5.1: Characteristics of test matrices

symmetric version. For more details about the other matrices we refer to Davis and Hu.

Running the metaheuristics on the test problems resulted in the bandwidths listed in Table 5.2. To take the randomness of the metaheuristics into account, for each combination of algorithm and matrix, both the average bandwidth (Bw) and the average required *cpu time* (Cpu) of 50 runs are listed. A (recent) computer has been used with an Intel Core i7 870 CPU @ 2.93GHz processor and 4.00 GB of RAM. Note that the cpu time is specific to this computer and implementation and is only listed to give an indication of the running time.

For both the hill climbing and simulated annealing the same way of determining the number of steps has been used. It was chosen based on numerical experimentation to find a good balance between running time and effectiveness. The number of steps was equal to 100 000 000 divided by the number of rows of the matrix. For the simulated annealing, the starting temperature was 10 and the stopping temperature was 0. The first run of simulated annealing used T_{slow} as cooling schedule and the second run used T_{fast} as cooling schedule.

Name	Hill climbing		Sim. ann. T_{slow}		Sim. ann. T_{fast}	
	Bw	*Cpu*	Bw	*Cpu*	Bw	*Cpu*
`m200`	52.1	*2.023*	37.3	*2.201*	43.3	*2.111*
`m200s`	50.3	*2.000*	33.4	*2.191*	39.0	*2.109*
`m500b80`	205.2	*2.541*	160.0	*2.553*	142.8	*2.584*
`m1000`	546.2	*1.619*	536.7	*1.648*	**535.4**	*1.638*
`m1000s`	547.4	*1.619*	539.1	*1.650*	**538.2**	*1.637*
`bayer03`	4191.8	*3.540*	3795.8	*3.642*	3798.8	*3.635*
`circuit_3`	5654.9	*4.749*	5429.1	*4.932*	5431.2	*4.940*
`extr1`	1133.6	*1.875*	942.3	*1.955*	943.5	*1.954*
`hydr1`	2915.1	*3.543*	2565.2	*3.731*	2567.2	*3.723*
`impcol_d`	92.7	*1.934*	44.5	*2.057*	52.6	*2.103*
`jan99jac020sc`	2364.2	*3.745*	2127.0	*3.843*	2126.4	*3.836*
`poli_large`	6240.9	*5.335*	5310.4	*5.464*	5315.6	*5.460*
`radfr1`	96.0	*1.830*	56.7	*1.894*	69.3	*1.907*
`rdist1`	1088.7	*3.070*	800.9	*3.164*	788.0	*3.145*
`sinc15`	5093.2	*4.447*	5071.9	*4.693*	5064.5	*4.698*
`Zhao2`	30369.9	*10.650*	29562.1	*10.856*	29565.7	*10.849*

Table 5.2: Results of metaheuristics (cpu is in seconds)

On average, the simulated annealing algorithms performed better than hill climbing for all matrices. For simulated annealing, the cooling schedule T_{slow} seems to perform similar to T_{fast}. However, for matrices with a small resulting bandwidth, T_{slow} seems to perform a bit better.

Applying the level set algorithms Cuthill-McKee, COBRA and COBRA with simulated annealing as post processing step resulted in the bandwidths as shown in Table 5.3. Again, for each combination of algorithm and matrix both the average bandwidth and the average cpu time of 50 runs on the same computer are listed.

Before running the Cuthill-McKee algorithm, the rows and columns of the matrix were sorted by increasing number of nonzero elements. A consequence of this preordering is that for each component of the graph representation, the algorithm will choose the starting vertex with the smallest nonzero degree. The COBRA algorithm was run with parameter $t = 10$, so ten different starting nodes were tried for each matrix. For the post processing with simulated annealing, T_{slow} was used as cooling schedule.

In general, the Cuthill-McKee algorithm performed better than simulated

Name	Cuthill-McKee		COBRA		COBRA + SA	
	Bw	*Cpu*	Bw	*Cpu*	Bw	*Cpu*
`m200`	62.8	*0.001*	49.1	*0.004*	**34.4**	*2.163*
`m200s`	50.9	*0.001*	40.9	*0.002*	**29.9**	*2.110*
`m500b80`	236.0	*0.009*	128.0	*0.052*	**82.6**	*2.584*
`m1000`	677.5	*0.009*	664.8	*0.085*	559.0	*1.752*
`m1000s`	677.2	*0.007*	664.6	*0.086*	559.1	*1.751*
`bayer03`	302.0	*0.300*	219.3	*3.472*	**194.0**	*6.427*
`circuit_3`	6061.1	*0.949*	2939.2	*13.511*	**2881.4**	*17.919*
`extr1`	67.0	*0.055*	59.6	*0.610*	**50.2**	*2.994*
`hydr1`	153.8	*0.185*	109.5	*2.171*	**97.3**	*4.990*
`impcol_d`	66.6	*0.001*	45.0	*0.017*	**31.4**	*1.819*
`jan99jac020sc`	1746.5	*0.299*	1304.2	*4.088*	**1222.4**	*6.814*
`poli_large`	6092.6	*1.539*	3853.3	*20.299*	**2925.4**	*26.344*
`radfr1`	72.6	*0.011*	51.5	*0.091*	**29.0**	*1.746*
`rdist1`	144.2	*0.123*	116.5	*1.316*	**70.3**	*3.165*
`sinc15`	4462.5	*0.934*	3294.5	*12.324*	**3164.0**	*15.996*
`Zhao2`	820.1	*7.236*	539.9	*83.539*	**538.8**	*95.637*

Table 5.3: Results of level set algorithms (cpu is in seconds)

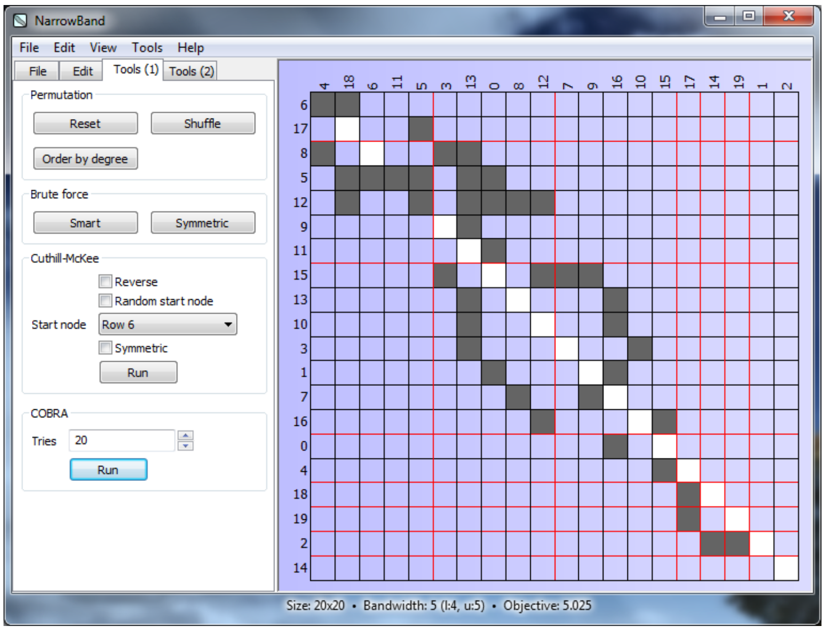
annealing for large matrices with a large number of nonzero elements. Otherwise, simulated annealing performed better. However, the running time of the Cuthill-McKee algorithm was a lot smaller than the running time of simulated annealing. For most matrices, COBRA resulted in a big improvement over the Cuthill-McKee algorithm at the cost of at least 10 times the running time (which follows from the parameter $t = 10$). Especially for larger matrices, such as `Zhao2`, it might be better to choose a smaller value for t.

Finally, the simulated annealing heuristic was most of the time able to improve the bandwidth even more. It may be noted that it reached a bandwidth of 83 for `m500b80`, which is very close to its assumed optimum of 80. The smallest bandwidth of a matrix of Tables 5.2 and 5.3 is typeset in bold. Clearly, COBRA combined with simulated annealing resulted in the best bandwidth for most test problems. However, it also had the largest running time.

While efficiency was kept in mind while implementing the algorithms, there might be ways to improve the running times of the algorithms. For instance, the matrix representation of the matrices was used as input. The metaheuristics acted upon the matrix representation and the level set algorithm first had to convert the matrix to the bipartite graph representation. Probably some improvement in running time is achievable by using the bipartite graph representation as input and by letting the metaheuristics act on this representation.

5.8 NarrowBand

All methods of this chapter have been implemented in C++ as a library. This made it possible to quickly write programs to test the various bandwidth reduction methods. Besides this, a *graphical user interface* (*GUI*, see Figure 5.18), called *NarrowBand*, has been created for this library to interactively modify matrices and permutations, save and load results, test different methods and algorithms and validate the results of this thesis.

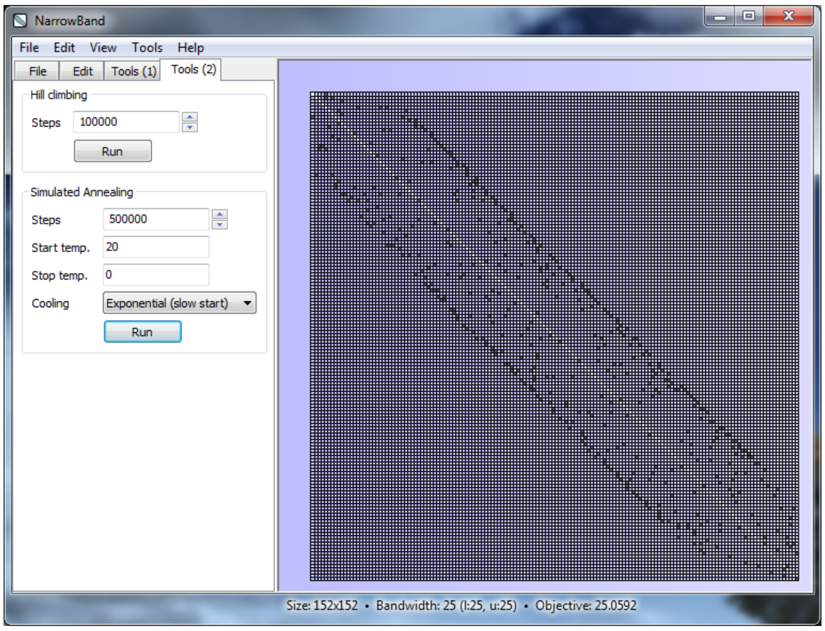


Start node: Row 6

Tries: 20

Size: 20x20 • Bandwidth: 5 (l:4, u:5) • Objective: 5.025

(a) NarrowBand after performing COBRA on a 20×20 matrix



Steps: 100000

Steps: 500000

Start temp.: 20

Stop temp.: 0

Cooling: Exponential (slow start)

Size: 152x152 • Bandwidth: 25 (l:25, u:25) • Objective: 25.0592

(b) NarrowBand after performing simulated annealing on a 152×152 matrix

Figure 5.18: Graphical user interface

6 Conclusions

This thesis covered some graph-theoretical aspects of constraint solving in the SST project. One important goal of SST framework is to generate a number of different designs of a product or machine given a model in the form of a system of equations. The algebraic form of the equations is unknown, but the structure of the system of equations is known in the form of a bipartite graph.

The Dulmage-Mendelsohn decomposition was used on the bipartite graph to split the graph in an under-, over- and well-constrained part. A new proof of the properties of this decomposition has been presented that described the decomposition in terms of maximum matchings. Also, an interactive method has been presented to generate a specific design given a model using the Dulmage-Mendelsohn decomposition and Broyden's method.

Besides considering the generation of designs in the SST project, the bandwidth reduction problem for unsymmetric matrices by permuting its rows and columns has been investigated. A proof has been presented for the NP-completeness of this problem using the bipartite graph representation of the problem. Several heuristics have been proposed to reduce the bandwidth of an unsymmetric matrix, including the level set algorithms Cuthill-McKee and COBRA and the metaheuristics hill climbing and simulated annealing.

All the methods to reduce the bandwidth of an unsymmetric matrix have been implemented in C++, including a graphical user interface. Computational results revealed that the COBRA algorithm combined with simulated annealing was the best way to reduce the bandwidth of most matrices considered in our numerical experiments.

6.1 Recommendations

Regarding the generation of designs in the SST project, several problems can be considered for further research. This thesis only investigated the equations and real variables within the model. Further research can focus on including inequalities and integer values into the model. That would be useful for the SST project. It may also be worthwhile to try to find ways to improve Broyden's method. Further it might be interesting to see what is possible within the SST project when the algebraic form of the system of equations (and the inequalities) of the model is known.

As to the reduction of the bandwidth of an unsymmetric matrix, probably several improvements are possible to both the level set algorithms and the metaheuristics in this thesis. Of particular interest would be the consequences of running the metaheuristics on the bipartite graph representation of the matrix. The possible gain in efficiency can be very useful for practical applications.

A Smart Synthesis Tools

IOP-IPCR Project 501

This appendix provides an overview of the Smart Synthesis Tools project. The information given in this appendix is also available on the internet: `http://www.opm.ctw.utwente.nl/research/design_engineering/Project%20Smart%20Synthesis%20Tools.doc/index.html`.

The Smart Synthesis Tools project aims to develop a next generation of CAD-systems. These new types of synthesis based computer tools help mechanical engineers to design solutions of higher quality in a significantly shorter time than is currently done.

It is a joined research effort of the University of Twente (CTW, EWI) and Delft University (3ME) together with four Dutch industries.

This project is funded by SenterNovem, an agency of the Dutch Ministry of Economic Affairs, in the framework of the IOP-IPCR program (Innovative Product Creation and Realization).

A.1 The partners

University of Twente (faculty CTW), contact `h.tragter@utwente.nl`

Technical University Delft (faculty 3ME), contact `t.tomiyama@wbmt.tudelft.nl`

University of Twente (faculty EWI), contact `g.still@utwente.nl`

Océ Technologies, Venlo

PANalytical, Almelo

Philips Domestic Appliances and Personal Care (DAPC), Drachten

Vanderlande Industries, Veghel

A.2 General

The objective is a further development of synthesis based design tools, of which several prototypes already have been build in Twente. Synthesis is seen in this context as the process of creating solutions from a set of (incomplete) specifications of the required behavior. The solutions are completely defined and optimal configured designs.

Experiences with the existing prototypes are very promising. They show that it is possible to generate optimal solutions for engineering problems, in significantly shortened time: up to ten times faster than with the current way of creating designs.

For a designer, the biggest gain can be achieved with the selection of a good concept. The research focuses on the development and integration of synthesis tools into a multidisciplinary design support system that can be applied at this concept level of design.

The tools will not, like a wishing well, invent new products, but they will assist engineers take the right decisions early in the process. They also will generate- and evaluate many solutions and help the engineer gain insight in the solution space.

A.3 Approach

The project encloses four subprograms, each of which is handled by one PhD student. The four subprograms together create the opening for future synthesis developments.

Domain Integration (Delft, 2ME): This subprogram will handle the design of mechatronic systems, which calls for a special kind of integration between mechanical, electronics, control systems and software design.

Structured design (Twente, CTW): For large problems, it helps to divide the total problem into smaller functional components to allow for tuned partial problem solving. Integrated solving on a global level has to be combined with detailed solving of separated details. An automated sub structuring of functions and integration of solutions is addressed in this subprogram.

Experience based synthesis (Twente, CTW): Incorporating existing design knowledge of experienced designers into synthesis tools will be used to increase the speed and quality of the solution finding process. This knowledge will be applied to the process of creating solution proposals, the process of parameter reduction, interpreting analysis results and to create feedback for optimization.

Large solution spaces (Twente, EWI): Efficient interactive 'navigation' techniques through high dimensional solution spaces are required in order to find the best design. The mathematical techniques for this have to be developed.

A bottom-up approach is used, where for each industrial partner a specific prototype tool will be developed. This ensures the generation of knowledge that has a broad applicability and that will create knowledge that can be used to build design tools for a whole range of industrial applications.

A.4 Innovation

The research has important innovations:

- The idea of a synthesis based design tool is itself a unique approach to the phenomena design support. In a synthesis tool, the designer can tell what he wants to achieve (by entering incomplete specifications). In a traditional tool you enter how you want to achieve it (the solution).
- Finding solutions in large design spaces, and the ability for a designer to navigate, and gain insight in his options for solutions is an important innovation which reaches far outside this project.
- The combination of a controlled generate-and-test algorithm to create solutions for design problems with a multi domain design system (horizontal integration) is unique.

A.5 Results

Knowledge will be collected to apply synthesis support for design problems with a higher complexity than the existing prototypes. The result will be threefold: there will be generic knowledge, a generic toolkit and prototype design systems.

The knowledge includes methods for development of synthesis tools, tool architectures and methods to characterize design problems and solution spaces.

With a combination of knowledge and generic software bundled in a toolkit, new industrial applications can be developed for specific areas of design.

Prototype tools will be developed for each of the four industrial partners. The tools applications are the design of mechatronic systems in a high volume printer, systems design of röntgen analysis equipment, cooling system for injection moulds and the design of luggage handling systems. All support the design process in the conceptual phase. The engineer is provided insight through the possibility to compare the many different solutions that are available.

B Tarjan's Algorithm

To find all strongly connected components of a graph one could use *Tarjan's Algorithm*, see (Tarjan, 1972). Algorithm B.1 shows a recursive version of Tarjan's Algorithm. Some practical applications require a non-recursive version of the algorithm, as shown in Algorithm B.2.

Algorithm B.1 Recursive version of Tarjan's Algorithm

Data: A graph $G := (V, E)$
Result: Strongly connected components printed out

```
index ← 0
$S ← \emptyset$   // $S$ is a stack
forall the $v$ in $V$ do
    if $v.index$ is undefined then
        tarjan($v$)

function tarjan($v$)
    $v$.index ← index
    $v$.lowlink ← index
    index ← index +1
    $S$.push($v$)
    forall the $(v, v')$ in $E$ do
        if $v'.index$ is undefined then
            tarjan($v'$)
            $v$.lowlink ← min($v.lowlink$, $v'.lowlink$)
        else if $v'$ is in $S$ then
            $v$.lowlink ← min($v.lowlink$, $v'.index$)
    if $v.lowlink$ is $v.index$ then
        print("SCC:")
        repeat
            $v'$ ← $S$.pop()
            print($v'$)
        until $v'$ is $v$
```

Algorithm B.2 Non-recursive version of Tarjan's Algorithm

Data: A graph $G := (V, E)$
Result: Strongly connected components printed out

```
index ← 0
$S \leftarrow \emptyset$   // $S$ is a stack
$to\_visit \leftarrow \emptyset$   // $to\_visit$ is a stack
forall the $v$ in $V$ do
    if $v.index$ is undefined then
        tarjan($v$)

function tarjan($v$)
    $to\_visit$.push($v$)
    while $to\_visit \neq \emptyset$ do
        $v \leftarrow to\_visit$.peek()
        if $v.index$ is undefined then
            $v$.index ← index
            $v$.lowlink ← index
            index ← index +1
            $S$.push($v$)
            forall the $(v, v')$ in $E$ do
                if $v'.index$ is undefined then
                    $v'$.parent ← $v$
                    $to\_visit$.push($v'$)
                else if $v'$ is in $S$ then
                    $v$.lowlink ← min($v.lowlink$, $v'.index$)
        else
            $to\_visit$.pop()
            if $v.parent$ is not undefined then
                $v$.parent.lowlink = min($v.parent.lowlink$, $v.lowlink$)
            if ($v.lowlink$ is $v.index$) and ($v$ is in $S$) then
                print("SCC:")
                repeat
                    $v' \leftarrow S$.pop()
                    print( $v'$)
                until $v'$ is $v$
```

Bibliography

S. Ait-Aoudia, Roland Jegou, and Dominique Michelucci. Reduction of constraint systems. *Proceedings of Compugraphics*, 1993.

C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92):pp. 577–593, 1965. ISSN 00255718.

C. G. Broyden, J. E. Dennis, and Jorge J. Moré. On the local and superlinear convergence of Quasi-Newton methods. *IMA Journal of Applied Mathematics*, 12(3):223–245, December 1973. ISSN 0272-4960. doi: 10.1093/imamat/12.3.223.

C.G. Broyden. On the discovery of the "good broyden" method. *Mathematical Programming*, 87:209–213, 2000. ISSN 0025-5610. 10.1007/s101070050111.

V. Cerný. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985. ISSN 0022-3239. 10.1007/BF00940812.

G. M. Del Corso and G. Manzini. Finding exact solutions to the bandwidth minimization problem. *Computing*, 62(3):189–203, 1999.

E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM. doi: http://doi.acm.org/10.1145/800195.805928.

T. A. Davis and Y. Hu. University of Florida sparse matrix collection. ACM Transactions on Mathematical Software (to appear). http://www.cise.ufl.edu/research/sparse/matrices.

J.E. Dennis and R.B. Schnabel. *A new derivation of symmetric positive definite secant updates*. Rice University, Dept. of Mathematical Sciences, 1980.

Gerhard W. Dueck and Janice Jeffs. A heuristic bandwidth reduction algorithm. *J. Combin. Math. Combin. Comput.*, 18:97–108, 1995. ISSN 0835-3026.

A. L. Dulmage and N. S. Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958. doi: 10.4153/CJM-1958-052-0.

A. L. Dulmage and N. S. Mendelsohn. A structure theory of bipartite graphs of finite exterior dimension. *Trans. Roy. Soc. Canada, Section*, (53): 1–13, 1959.

A. L. Dulmage and N. S. Mendelsohn. *Graph Theory and Theoretical Physics*, chapter Graphs and matrices, pages 167–277. Academic Press, London and New York, 1967.

A. Esposito, M. S. Fiorenzo Catalano, F. Malucelli, and L. Tarricone. A new matrix bandwidth reduction algorithm. *Operations Research Letters*, 23 (3-5):99–107, 1998a. ISSN 0167-6377. doi: DOI:10.1016/S0167-6377(98) 00040-6.

Alessandra Esposito, Federico Malucelli, and Luciano Tarricone. Bandwidth and profile reduction of sparse matrices: An experimental comparison of new heuristics. In *Algorithms and Experiments (ALEX'98)*, Trento, 1998b.

Ulrich Faigle, George Still, and Walter Kern. *Algorithmic principles of mathematical programming*, chapter Unconstrained optimization, page 256. Kluwer Academic Publishers, 2002.

David M. Gay. Some convergence properties of broyden's method. *SIAM Journal on Numerical Analysis*, 16(4):623–630, August 1979.

P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10:26, 1935.

John E. Hopcroft and Richard M. Karp. An $n^{(5/2)}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, December 1973.

H.T. Jongen, P. Jonker, and F. Twilt. *Nonlinear optimization in finite dimensions: Morse theory, Chebyshev approximation, transversality, flows, parametric aspects*, volume 47. Kluwer Academic Publishers, 2000.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):pp. 671–680, 1983. ISSN 00368075.

John G. Lewis. Algorithm 582: The gibbs-poole-stockmeyer and gibbs-king algorithms for reordering sparse matrices. *ACM Trans. Math. Softw.*, 8: 190–194, June 1982. ISSN 0098-3500. doi: http://doi.acm.org/10.1145/355993.355999.

L. Lovász and M. D. Plummer. *Matching theory*, chapter 3.2, pages 93–102. Elsevier Science Publishers B.V., 1986.

J.-C. Luo. Algorithms for reducing the bandwidth and profile of a sparse matrix. *Computers & Structures*, 44(3):535–548, 1992. ISSN 0045-7949. doi: DOI:10.1016/0045-7949(92)90386-E.

Rafael Martí, Manuel Laguna, Fred Glover, and Vicente Campos. Reducing the bandwidth of a sparse matrix with tabu search. *European J. Oper. Res.*, 135(2):450–459, 2001. ISSN 0377-2217. doi: 10.1016/S0377-2217(00)00325-8.

R. Martin and J. Wilkinson. Solution of symmetric and unsymmetric band equations and the calculation of eigenvectors of band matrices. *Numerische Mathematik*, 9:279–301, 1967. ISSN 0029-599X. 10.1007/BF02162421.

Ch. H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16:263–270, 1976.

Estefanía Piñana, Isaac Plana, Vicente Campos, and Rafael Marti. GRASP and path relinking for the matrix bandwidth minimization. *European J. Oper. Res.*, 153(1):200–210, 2004. ISSN 0377-2217. doi: 10.1016/S0377-2217(02)00715-4.

P. Pop and O. Matei. An improved heuristic for the bandwidth minimization based on genetic programming. In Emilio Corchado, Marek Kurzynski, and Michal Wozniak, editors, *Hybrid Artificial Intelligent Systems*, volume 6679 of *Lecture Notes in Computer Science*, pages 67–74. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-21222-2-9.

Alex Pothen. *Sparse null bases and marriage theorems*. PhD thesis, Ithaca, NY, USA, 1984. AAI8415425.

J. K. Reid and J. A. Scott. Reducing the total bandwidth of a sparse unsymmetric matrix. *SIAM Journal on Matrix Analysis and Applications*, 28(3): 805–821, 2006. doi: 10.1137/050629938.

Eduardo Rodriguez-Tello, Jin-Kao Hao, and Jose Torres-Jimenez. An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research*, 185(3):1319 – 1335, 2008. ISSN 0377-2217. doi: DOI:10.1016/j.ejor.2005.12.052.

SW Sloan. A fortran program for profile and wavefront reduction. *International Journal for Numerical Methods in Engineering*, 28(11):2651–2679, 1989. ISSN 1097-0207.

Georg Still, Walter Kern, Jaap Koelewijn, and Matthijs Bomhoff. Consistency of a system of equations: What does that mean? Technical Report 1929, Department of Applied Mathematics, University of Twente, 2010.

Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

Walter Unger. The complexity of the approximation of the bandwidth problem. In *Foundations of Computer Science (FOCS)*, pages 82–91, 1998.

Index