Spring 2013

Graph-theoretic studies of combinatorial optimization problems

Seyed Mohammad S. Mirghorbani Nokandeh
University of Iowa

GRAPH-THEORETIC STUDIES OF COMBINATORIAL OPTIMIZATION PROBLEMS

by

Seyed Mohammad S. Mirghorbani Nokandeh

An Abstract

Of a thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Industrial Engineering
in the Graduate College of
The University of Iowa

May 2013

Thesis Supervisor: Prof. Pavlo Krokhmal

ABSTRACT

Leonhard Euler introduced the concept of Graph Theory in his paper about the seven bridges of Konigsberg published in 1736. It is the study of pair-wise relationships between objects. Each object is represented using a vertex, and in case of a relationship between a pair of vertices, they will be connected using an edge.

In this dissertation, graph theory is used to study several important combinatorial optimization problems. In chapter 2, we study the multi-dimensional assignment problem using its underlying hypergraphs. It will be shown how the MAP can be represented by a k-partite graph and how any solution to MAP is associated to a k-clique in the respective k-partite graph. Two heuristics are proposed to solve the MAP and computational studies are performed to compare the performance of the proposed methods with exact solutions. On the heels of chapter 2, a new branch-and-bound method is proposed to solve the problem of finding all k-cliques in a k-partite graph in chapter 3. The new method utilizes bitsets as the datastructure to represent graph data. A new pruning method is introduced in BitCLQ, and CPU instructions are used to improve the performance of the branch-and-bound method. BitCLQ gains up to 300% speed up over existing methods. In chapter 4, two new heuristics to solve decomposable cost MAP's are proposed. The proposed heuristic are based on the partitioning of the underlying graph representing the MAP. In the first heuristic method, MAP's are partitioned into several smaller MAP's with the same dimensial-

ity and smaller cardinality. The second heuristic works in the same fashion. But instead of partitioning the graph along the elements, graphs are divided into smaller graphs with the same cardinality but smaller dimensionality. The heuristics are then used in exact branch and bound methods and numerical comparison of the resulting method is provided. Maximum Clique problem entails finding the size of the largest clique contained in a graph. General branch-and-bound methods to solve MCQ use graph coloring to find an upper bound on the size of the maximum clique. In chapter 5, a branch and bound algorithm is proposed for the maximum clique problem. that is based on the method of 5. Chapter 6 contains an application of a graph theory in solving a risk management problem. The mixed-integer mathematical model to formulate a risk-based network is provided. It will be shown that an optimal solution of the model is a maximal clique in the underlying graph representing the network.

Abstract Approved: _____
Thesis Supervisor

_____
Title and Department

_____
Date

GRAPH-THEORETIC STUDIES OF COMBINATORIAL OPTIMIZATION PROBLEMS

by

Seyed Mohammad S. Mirghorbani Nokandeh

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Industrial Engineering
in the Graduate College of
The University of Iowa

May 2013

Thesis Supervisor: Prof. Pavlo Krokhmal

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Seyed Mohammad S. Mirghorbani Nokandeh

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Industrial Engineering at the May 2013 graduation.

Thesis Committee: _____
Pavlo Krokhmal, Thesis Supervisor

_____
Mona Garvin

_____
Andrew Kusiak

_____
Yong Chen

_____
Geb Thomas

ABSTRACT

Leonhard Euler introduced the concept of Graph Theory in his paper about the seven bridges of Konigsberg published in 1736. It is the study of pair-wise relationships between objects. Each object is represented using a vertex, and in case of a relationship between a pair of vertices, they will be connected using an edge.

In this dissertation, graph theory is used to study several important combinatorial optimization problems. In chapter 2, we study the multi-dimensional assignment problem using its underlying hypergraphs. It will be shown how the MAP can be represented by a k-partite graph and how any solution to MAP is associated to a k-clique in the respective k-partite graph. Two heuristics are proposed to solve the MAP and computational studies are performed to compare the performance of the proposed methods with exact solutions. On the heels of chapter 2, a new branch-and-bound method is proposed to solve the problem of finding all k-cliques in a k-partite graph in chapter 3. The new method utilizes bitsets as the datastructure to represent graph data. A new pruning method is introduced in BitCLQ, and CPU instructions are used to improve the performance of the branch-and-bound method. BitCLQ gains up to 300% speed up over existing methods. In chapter 4, two new heuristics to solve decomposable cost MAP's are proposed. The proposed heuristic are based on the partitioning of the underlying graph representing the MAP. In the first heuristic method, MAP's are partitioned into several smaller MAP's with the same dimensial-

ity and smaller cardinality. The second heuristic works in the same fashion. But instead of partitioning the graph along the elements, graphs are divided into smaller graphs with the same cardinality but smaller dimensionality. The heuristics are then used in exact branch and bound methods and numerical comparison of the resulting method is provided. Maximum Clique problem entails finding the size of the largest clique contained in a graph. General branch-and-bound methods to solve MCQ use graph coloring to find an upper bound on the size of the maximum clique. In chapter 5, a branch and bound algorithm is proposed for the maximum clique problem. that is based on the method of 5. Chapter 6 contains an application of a graph theory in solving a risk management problem. The mixed-integer mathematical model to formulate a risk-based network is provided. It will be shown that an optimal solution of the model is a maximal clique in the underlying graph representing the network.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

CHAPTER 1
INTRODUCTION

1.1 Optimization Problems

Many real life problems entail determining the best configuration for a set of decision variables to achieve an optimum value for some criteria. Over the past decade, different types of such problems have emerged and corresponding techniques for solving them are developed. Many optimization problems are concerned with optimizing a function f over a finite set X of d-tuples of integers. If f is linear and X is defined by a finite number of linear inequality constraints with integer coefficients, the problem in hand is called annnnn *integer programming*.

Combinatorial optimization is a subset of discrete mathematics with its roots in combinatorics, operations research, and theoretical computer science. The propelling force behind the ongoing research in this field is that many practical problems can be formulated as combinatorial optimization problems. Most combinatorial optimization problems deal with problems that can be formulated as integer programs, but have an underlying combinatorial structure that makes it possible to develop special algorithms that are expected to be more efficient than general integer programming methods. Many of these problems can be represented on a directed or undirected graph $G(V, E)$, and a function f defined on the finite node set V or the edge set E or the union of the vertex set and the edge set, which takes real values. Some of the most important problems studies in this area include: the vehicle routing problem,

traveling salesman problem, minimum spanning tree, matching problem, scheduling problem, n-queen problem, assignment problem and weapon-target assignment problem.

In this dissertation we will study two important combinatorial optimization problems, namely, the multi-dimensional assignment problem, and the maximum clique problem. The multi-dimensional assignment problems are an extension of the two-dimensional assignment problems. In the first part of this chapter, we will introduce several types of assignment problems and some of their properties. In the second part of the chapter, the maximum clique problem, related definitions, and the formulation of the problem will be provided.

1.2 Assignment Problems

The simplest form of assignment problems is the linear assignment problem (LAP). LAP is one of the fundamental models in operations research, computer science, and discrete mathematics. In its most familiar interpretation, it answers the question of finding an assignment of n workers to n jobs that has the lowest total cost, if the cost of assigning worker i to task j equals c_{ij} . Apart from the straightforward applications, such as personnel assignment problems, the LAP frequently arises as a part of other optimization problems, such as quadratic assignment problem, multidimensional assignment problem, traveling salesman problem, etc. Other applications of the LAP, including earth-satellite systems with TDMA protocol, and tracking objects in space are considered in [14] and [12]; for a more comprehensive discussion of

the applications of the LAP, refer to, e.g., [16].

A mathematical programming formulation of the LAP reads as

$$\begin{aligned} L_n^* = \min_{x_{ij}\in\{0,1\}} & \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_{ij} \\ \text{s. t.} & \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,\ldots,n, \\ & \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,\ldots,n, \end{aligned} \tag{1.1}$$

where it is well known that the integrality of variables x_{ij} can be relaxed: $0 \le x_{ij} \le 1$, leading to an integer programming or linear programming formulation of LAP. In the graph theoretical setting, the optimal solution to LAP corresponds to the minimum-cost perfect matching[1] in an edge weighted bipartite graph. An LAP can be solved in polynomial time; the best worst-case complexity for LAP is $\mathcal{O}(n^3)$, obtained from the primal-dual Hungarian method [42].

1.2.1 Multidimensional Assignment Problem

The multidimensional assignment problem (MAP) is a higher dimensional version of the (two-dimensional) LAP. For example, a 3-dimensional assignment problem can be interpreted as finding an optimal assignment of n jobs to n workers on n machines. In general, given d sets each of n elements, the objective is to find an optimal assignment of the elements of each set of n d-tuples, such that the total cost of the d-tuples is minimized. The axial d-dimensional assignment problem can be formulated as an integer programming problem with n^d binary variables and $n \times d$

[1] A *matching* in a graph is a set of edges without common vertices. A *perfect matching* is a matching that covers all the vertices in a graph.

constraints. Multi-dimensional assignment problems are furthered studied in chapters 2 and 4.

1.2.2 Quadratic Assignment Problems

Let $\mathbf{A} = (a_{ij})$ and $\mathbf{B} = (b_{ij})$ be two square $n \times n$ matrices. Then the quadratic assignment problem (QAP) as introduced by Koopmans and Beckmann [37] can be formulated as:

$$\begin{aligned} Q_n = \min_{x_{ij} \in \{0,1\}} & \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} a_{ij} b_{kl} x_{ik} x_{jl} \\ \text{s. t.} & \sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \ldots, n, \\ & \sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n, \end{aligned} \tag{1.2}$$

QAP was first formulated in the context of facility layout, with $\mathbf{A}$ being the matrix of distances between sites, and $\mathbf{B}$ the flow of goods. Of a comprehensive survey of the applications of QAP, please refer to [13, 54, 55, 15]. QAP is known to be NP-hard and non-approximable [64]. Formally, an approximation algorithm for a minimization problem is called ϵ-approximate[2] if for every input, the algorithm finds a solution whose objective function is at most ϵ times the optimum. A problem is called non-approximable if it is proved that unless $P = NP$, there is no ϵ-approximate algorithm to solve the problem. For an overview of the recent advances in exact and heuristic algorithms for the QAP, please refer to [15, 3, 48].

[2] ϵ is called the performance ratio of the algorithm.

1.2.3 Bottleneck Assignment Problems

The bottleneck assignment problems are closely related to the assignment problems with sum objective; if the latter minimize the total cost of all assignment, then in the corresponding bottleneck problem, the cost of the most expensive assignment is minimized.

1.2.3.1 Linear Bottleneck Assignment Problem

The linear bottleneck assignment problem, in the graph-theoric interpretation, tries to find the perfect matching in the weighted bipartite graph that minimizes the maximum weight of all matched edges. The mathematical formulation of the linear bottleneck assignment problem reads as:

$$\begin{aligned} Z_n = \min_{x_{ij}\in\{0,1\}} \quad & \max_{i,j} \quad c_{ij}x_{ik}x_{jl} \\ & \text{s. t.} \quad \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1,\ldots,n, \\ & \qquad \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1,\ldots,n, \end{aligned} \tag{1.3}$$

For the applications of the bottleneck assignment problem, please refer to [25, 6, 17].

1.2.3.2 Quadratic Bottleneck Assignment Problem

First introduced by [68] in the context of backboard wiring problem, the quadratic bottleneck assignment problem can be obtained by replacing the objective function in (1.3) with:

$$Z_n = \min_{\pi \in \Pi_n} \max_{i,j} a_{i,j} b_{\pi(i),\pi(j)}, \tag{1.4}$$

This problem is proved to be NP-hard

1.2.4 Properties of the optimal random assignments problems

First attempts to compute the expected optimal value of random LAP's (with i.i.d cost coefficients, c_{ij}) were done in [43] and [21]. In[74] an upper bound for the expected cost of (1.1) when the cost coefficients c_{ij} are iid uniform [0,1] random variables is established:

$$E[L_n] \leq 3.$$

This result was improved in [34] utilizing LP duality: $E[L_n] \leq 2$. Later on, a tighter upper bound of 1.94 was established in [20] via application of an assignment algorithm to large cost matrices whose entries are iid exponential with mean 1.

With an assumption of iid uniform [0,1] assignment costs, the first lower bound on the value of an optimal assignment in (1.1) was developed in [44] using the dual LP formulation and evaluating the dual objective after row and column reductions due to the Hungarian method: $E[L_n] \geq 1.368$. This result was later improved by the dual heuristic proposed in [52] showing that $E[L_n] \geq 1.51$.

1.3 Maximum Clique Problem

One of the classical problems in the combinatorial optimization domain, the maximum clique problem, has important applications in different areas such as infor-

mation retrieval, experimental design, signal transmission, and computer vision. In the remainder of this chapter, we provide the formulations and priliminary definitions for the maximum clique problem.

1.3.1 Definitions

Given an undirected graph $G(V, E)$, where $V = \{1, 2, \ldots, n\}$ is the vertex set and $E \subseteq V \times V$ is the edge set of G, a *clique* is defined as a complete subgraph of G; i.e. a set of nodes that are pair-wise adjacent. $A_G = (a_{ij})_{(i,j)\in V\times V}$, where $a_{ij} = 1$ if $(v_i, v_j) \in E$ and $a_{ij} = 0$ if $(v_i, v_j) \notin E$, is called the *adjacency matrix* of G. In case of a weighted graph, a positive weight w_i is associated with vertex i. Weights can be associated to edges as well. The *neighborhood* of node v_i, $N(v_i)$ is defined as the set of nodes that are adjacent to it:

$$N(v_i) = \{v_j | (v_i, v_j) \in E\}.$$

The complement of graph G, denoted by $\overline{G}(V, \overline{E})$ is a graph with vertex set V and the edge set $\overline{E} = \{(v_i, v_j) | v_i, v_j \in V, v_i \neq v_j, \text{ and } (v_i, v_j) \neq E\}$. For a subgraph $S \subseteq V$, weigh of S is denoted by $W(S)$ and is equal to $\sum_{i\in S} w_i$. Also $G(S) = (S, E \cap (S \times S))$ is called the subgraph induced by S.

A graph $G(V, E)$ is called *complete* if all its vertices are pair-wise adjacent. A clique C is complete subgraph of G, i.e. G(C) is a complete graph. A clique is called *maximal* if it is not a subset of another clique. The largest clique in a graph is called the *maximum clique*. The size of the maximum clique in a graph is called the *clique*

number of G and is denoted by $\omega(G)$.

The maximum clique problem seeks to find the clique number in a given graph. The maximum weight clique problem seeks to find the cliques of maximum weight. Consequently, the *weighted clique number* is the sum of the weights of the edges in the maximum weight clique and is denoted by $\omega(G, w)$:

$$\omega(G) = \max\{|S| : S \text{ is a clique in } G\}$$

$$\omega(G, w) = \max\{W(S) : S \text{ is a clique in } G\}$$

An *independent set*, also known as *stable set* or *vertex packing* is a set of nodes that are pair-wise non-adjacent. The maximum independent set problem seeks to find the independent set of the maximum cardinality in a given graph, the size of which is called the *stability number* of G and is denoted by $\alpha(G)$. Similarly to the maximum weight clique problem, one can define the maximum weight independent set.

A *vertex cover* in graph G is a set of vertices such that every edge $(v_i, v_j) \in E$ has at least one neighbor in the subset. The minimum vertex cover problem seeks to find a vertex cover of minimum cardinality.

It is easy to show that the following statements are equivalent concerning any $S \subset V$ [7]:

1. S is the vertex set of a maximum clique in G,
2. S is a maximum vertex packing in $\overline{G}$,
3. $V \setminus S$ is a minimum vertex cover in $\overline{G}$

As a result, the following three problems are equivalent:

1. Finding a maximum clique in G,
2. Finding a maximum vertex packing in $\overline{G}$
3. Finding a minimum vertex cover in $\overline{G}$.

They are also known to be NP-complete.

1.3.2 Formulations

The maximum clique problem has several formulations as an integer programming problem or as a continuous nonconvex optimization problem. The simplest formulation of the maximum clique problem is based on edge formulation:

$$\begin{array}{rlll} \omega(G) = & \max & \displaystyle\sum_{i=1}^{n} w_i x_i & \\ & \text{s. t.} & x_i + x_j \leq 1 & \forall (i,j) \in \overline{E} \\ & & x_i \in \{0,1\} & i = 1, \ldots, n, \end{array} \tag{1.5}$$

The polyhedral properties of (1.5) is studied in [50] and [51]:

Theorem 1.3.1. *Lex x be an optimum $(0, \frac{1}{2}, 1)$-valued solution to the linear relaxation of (1.5), and let $P = \{j | x_j = 1\}$. Then there exists an optimum solution x^* to (1.5), such that $x_j^* = 1$, $\forall j \in P$.*

This theorem can be used in enumerative algorithms. However, in most cases, few variables have integer values in an optimal solution to the linera relaxation of (1.5) and the gap between the relaxation and the optimal solution of (1.5) is usually too large.

An alternative formulation for the maximum clique problem can be made based on the maximal independet sets contained in G:

$$\begin{aligned} \omega(G) = \quad & \max \quad \sum_{i=1}^{n} w_i x_i \\ & \text{s. t.} \quad \sum_{i \in S} x_i \leq 1 \quad \forall S \in \mathcal{S} \\ & \qquad x_i \in \{0,1\} \quad i = 1, \ldots, n, \end{aligned} \tag{1.6}$$

where $\mathcal{S}$ is the set of all maximal independent sets of G. Although the relaxation to (1.6) produces a smaller gap compared to (1.5) with the optimal value, it has exponential number of constraints. The relaxation to (1.6) is actually proved to be NP-hard on general graphs. It can, however, be solved in polynomial time for perfect graphs, and under such cases, the optimal solution takes integer values [28].

The following theorem shows a zero-one quadratic formulation of the maximum clique problem:

Theorem 1.3.2. *The maximum clique problem is equivalent to the following global quadratic zero-one problem:*

$$\begin{aligned} \min f(x) = \quad & x^T A x \\ s.\ t. \quad & x \in \{0,1\}^n \quad \text{where } A = A_{\overline{G}} - \mathbf{I} \end{aligned} \tag{1.7}$$

If x^* solves (1.7), then the set $C = t(x^*)$ is a maximum clique of G with $|C| = -z = -f(x^*)$.

In theorem 1.3.2, t is a transformation from $\{0,1\}^n$ to 2^V:

$$t(x) = \{i \in V : x_i = 1\}, \forall x \in \{0,1\}^n,$$

and $\mathbf{I}$ is an $n \times n$ identity matrix.

For a complete survey on the maximum clique problem, formulations, exact and heuristic methods, and application, the reader is referred to [10].

In this dissertation, two important combinatorial problems are studied: the multi-dimensional assignment problem, and the maximum clique problem. In particular, in chapter 2, we propose two heuristics for the random multi-dimensional assignment problem and illustrate that the solution obtained from the heuristics is convergent to the optimal solution of the problem. Chapter 3 studies a particular case of the maximum clique problem, enumerating the k-cliques, in k-partite graphs. Chapter 5 explains a new method for the maximum clique problem in general graphs, and finally chapter 4 describes a new heuristic based on graph partitioning for the multi-dimensional assignment problem.

CHAPTER 2
COMPUTATIONAL STUDIES OF RANDOMIZED MULTIDIMENSIONAL ASSIGNMENT PROBLEMS

2.1 Introduction

In the simplest form of the assignment problem, two sets V and W with size $|V| = |W| = n$ are given. The goal is to find a permutation of the elements of W, $\pi = (j_1, j_2, \ldots, j_n)$, where the i^{th} element of V is assigned to the element $j_i = \pi(i)$ from W in such a way that the cost function $\sum_{i=1}^{n} a_{i\pi(i)}$ is minimized. Here, a_{ij} is the cost of assigning element i of V to the element j of W. This problem is widely known as the classical Linear Assignment Problem (LAP). The LAP can be represented by a complete weighted bipartite graph $G = (V, W; E)$, with node sets V and W, where $|V| = |W| = n$ and weight a_{ij} for the edge $(v_i, w_j) \in E$ (Fig. 3.1), such that an optimal solution for LAP corresponds to a minimum-weight matching in the bipartite graph G. The LAP is well known to be polynomially solvable in $O(n^3)$ time using the celebrated Hungarian method [42]. A mathematical programming formulation of the LAP reads as

$$
\begin{aligned}
L_n^* = \min_{x_{ij} \in \{0,1\}} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_{ij} \\
\text{s. t.} \quad & \sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \ldots, n, \\
& \sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1, \ldots, n,
\end{aligned}
\tag{2.1}
$$

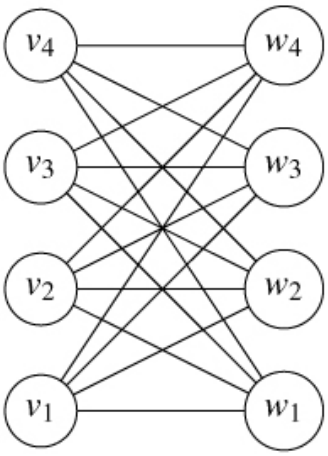


Figure 2.1: The underlying bi-partite graph for an assignment problem with $n = 4$

where it is well known that the integrality of variables x_{ij} can be relaxed: $0 \leq x_{ij} \leq 1$. The LAP also admits the following permutation based formulation:

$$\min_{\pi \in \Pi} \sum_{i=1}^{n} a_{i\pi(i)}, \tag{2.2}$$

where Π is the set of all permutations of the set $\{1, \ldots, n\}$.

Multidimensional extensions of the bipartite graph matching problems, such as the LAP, Quadratic Assignment Problem (QAP), and so on, can be presented in the framework of *hypergraph matching problems*.

A *hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, also called a *set system*, is a generalization of the graph concept, where a *hyperedge* may connect two or more vertices from the set $\mathcal{V}$:

$$\mathcal{E} = \{e \subset \mathcal{V} \mid |e| \geq 2\}, \tag{2.3}$$

A hypergraph is called k-*uniform* if all its hyperedges have the size k:

$$\mathcal{E} = \{e \in \mathcal{V} \mid |e| = k\}.$$

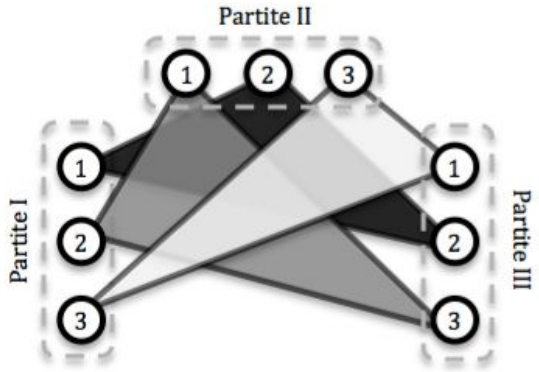


Figure 2.2: A perfect matching in a 3-partite 3-uniform hypergraph.

Observe that a regular graph is a 2-uniform hypergraph. A subset $\mathcal{V}' \subset \mathcal{V}$ of vertices is called *independent* if the vertices in $\mathcal{V}'$ do not share any edges; if $\mathcal{V}$ can be partitioned into d independent subsets, $\mathcal{V} = \cup_{k=1}^{d} \mathcal{V}_k$, then $\mathcal{V}$ is called d-partite.

Let $\mathcal{H}_{d|n}$ be a complete d-partite n-uniform hypergraph, where each independent set $\mathcal{V}_k$ has n vertices. Then $\left|\mathcal{V}(\mathcal{H}_{d|n})\right| = n \times d$, and the total number of hyperedges is equal to n^d. A *perfect matching* μ on $\mathcal{H}_{d|n}$ is formed by a set of n hyperedges that do not share any vertices:

$$\mu = \left\{ \{e_1, \ldots, e_n\} \;\middle|\; e_i \in \mathcal{E},\;\; e_i \cap e_j = \emptyset,\;\; i, j \in \{1, \ldots, n\}, i \neq j \right\}.$$

Figure 2.2 shows a perfect matching in a 3-partite 3-uniform hypergraph.

If the cost of hypergraph matching μ is given by function $\Phi(\mu)$, the general combinatorial optimization problem on hypergraph matchings can be stated as

$$\min \left\{ \Phi(\mu) \;\middle|\; \mu \in \mathcal{M}(\mathcal{H}_{d|n}) \right\}, \tag{2.4}$$

where $\mathcal{M}(\mathcal{H}_{d|n})$ is the set of all perfect matchings on $\mathcal{H}_{d|n}$.

The mathematical programming formulation of the hypergraph matching problem (2.4) is also generally known as *multidimensional assignment problem (MAP)*. To derive the mathematical programming formulation of (2.4), note that according to the definition of $\mathcal{H}_{d|n}$, each of its hyperedges contains exactly one vertex from each of the independent sets $\mathcal{V}_1, \ldots, \mathcal{V}_d$ and therefore can be represented as a vector $(i_1, \ldots, i_d) \in \{1, \ldots, n\}^d$, where, with abuse of notation, the set $\{1, \ldots, n\}$ is used to label the nodes of each independent subset $\mathcal{V}_k$. Then, the set $\mathcal{M}(\mathcal{H}_{d|n})$ of perfect matchings on $\mathcal{H}_{d|n}$ can be represented in a mathematical programming form as

$$\mathcal{M}(\mathcal{H}_{d|n}) = \left\{ x \in \{0,1\}^{n^d} \;\middle|\; \sum_{\substack{i_k \in \{1,\ldots,n\} \\ k \in \{1,\ldots,d\} \setminus \{r\}}} x_{i_1 \cdots i_d} = 1, \;\; i_r \in \{1, \ldots, n\}, \; r \in \{1, \ldots, d\} \right\}, \tag{2.5}$$

where $x_{i_1 \cdots i_d} = 1$ if the hyperedge $(i_1, \ldots, i_d)$ is included in the matching, and $x_{i_1 \cdots i_d} = 0$ otherwise.

Depending on the particular form of Φ, a number of combinatorial optimization problems on hypergraph matchings can be formulated. For instance, if the cost function Φ in (2.4) is defined as a linear form over the variables $x_{i_1 \cdots i_d}$,

$$\Phi(x) = \sum_{i_1=1}^{n} \cdots \sum_{i_d=1}^{n} \phi_{i_1 \cdots i_d} \, x_{i_1 \cdots i_d}, \tag{2.6}$$

one obtains the so-called linear multidimensional assignment problem (LMAP):

$$
\begin{aligned}
Z^*_{d,n} = \min_{x\in\{0,1\}^{n^d}} \quad & \sum_{i_1=1}^{n}\cdots\sum_{i_d=1}^{n}\phi_{i_1\cdots i_d}x_{i_1\cdots i_d} & & \\
\text{s. t.} \quad & \sum_{i_2=1}^{n}\cdots\sum_{i_d=1}^{n}x_{i_1\cdots i_d}=1, & & i_1=1,\ldots,n,\\
& \sum_{i_1=1}^{n}\cdots\sum_{i_{k-1}=1}^{n}\sum_{i_{k+1}=1}^{n}\cdots\sum_{i_d=1}^{n}x_{i_1\cdots i_d}=1, & & i_k=1,\ldots,n,\\
& & & k=2,\ldots,d-1,\\
& \sum_{i_1=1}^{n}\cdots\sum_{i_{d-1}=1}^{n}x_{i_1\cdots i_d}=1, & & i_d=1,\ldots,n.
\end{aligned}
\tag{2.7}
$$

Clearly, a special case of (2.7) with $d = 2$ is nothing else but the classical LAP (2.1). The *dimensionality* parameter d in (2.7) stands for the number of "dimensions" of the problem, or sets of elements that need to be assigned to each other, while the parameter n is known as the *cardinality* parameter.

If the cost of the matching on hypergraph $\mathcal{H}_{d|n}$ is defined as the cost of the most expensive hyperedge in the matching, i.e., the cost function $\Phi(x)$ has the form

$$\Phi(x) = \max_{i_1,\ldots,i_d\in\{1,\ldots,n\}} \phi_{i_1\cdots i_d}\, x_{i_1\cdots i_d},$$

we obtain the multidimensional assignment problem with bottleneck objective (BMAP):

$$
\begin{aligned}
W^*_{d,n} = \min_{x\in\{0,1\}^{n^d}} \quad & \max_{i_1,\ldots,i_d\in\{1,\ldots,n\}} \phi_{i_1\cdots i_d}\, x_{i_1\cdots i_d} & & \\
\text{s. t.} \quad & \sum_{i_2=1}^{n}\cdots\sum_{i_d=1}^{n}x_{i_1\cdots i_d}=1, & & i_1=1,\ldots,n,\\
& \sum_{i_1=1}^{n}\cdots\sum_{i_{k-1}=1}^{n}\sum_{i_{k+1}=1}^{n}\cdots\sum_{i_d=1}^{n}x_{i_1\cdots i_d}=1, & & i_k=1,\ldots,n,\\
& & & k=2,\ldots,d-1,\\
& \sum_{i_1=1}^{n}\cdots\sum_{i_{d-1}=1}^{n}x_{i_1\cdots i_d}=1, & & i_d=1,\ldots,n.
\end{aligned}
\tag{2.8}
$$

Similarly, taking the hypergraph matching cost function Φ in (2.4) as a quadratic

form over $x \in \{0,1\}^{n^d}$,

$$\Phi(x) = \sum_{i_1=1}^{n} \cdots \sum_{i_d=1}^{n} \sum_{j_1=1}^{n} \cdots \sum_{j_d=1}^{n} \phi_{i_1 \cdots i_d j_1 \cdots j_d} \, x_{i_1 \cdots i_d} \, x_{j_1 \cdots j_d}, \tag{2.9}$$

we arrive at the quadratic multidimensional assignment problem (QMAP), which represents a higher-dimensional generalization of the classical QAP.

The LMAP was first introduced by Pierskalla [57], and has found applications in the areas of data association, sensor fusion, multi-sensor multi-target tracking, peer-to-peer refueling of space satellites, etc; for a detailed discussion of the applications of the LMAP, see, e.g., [15, 16]. In [8], a two step method based on bipartite and multidimensional matching problem is proposed to solve the roots of a system of polynomial equations that avoids possible degeneracies and multiple roots encountered in some conventional methods. MAP is used in the course timetabling problem, where the goal is to assign students and teachers to classes and time slots [18]. In [1] a composite neighborhood structure with a randomized iterative improvement algorithm for the timetabling problem with a set of hard and soft constraints is proposed. An application of MAP in the scheduling of sport competitions that take place in different venues is studied in [72]. The characteristic of this study is that venues, that can involve playing fields, courts, or drill stations, are considered as part of the scheduling process. In [58] a Lagrangian relaxation based algorithms is proposed for the multi-target/multi-sensor tracking problem, where multiple sensors are used to identify targets and estimate their states. To accurately achieve this goal, the data association problem which is an NP-hard problem should be solved to partition observations into tracks and false alarms. A general class of these data association prob-

lems can be formulated as a multi-dimensional assignment problem with a Bayesian estimation as the objective function. the optimal solution yields the maximum a posteriori estimate. A special case of multiple-target tracking problem is studied in [60] to track the flight paths of charged elementary particles near their primary point of interaction. The 3-dimensional assignment problem is used in [22] to formulate a peer-to-peer (P2P) satellite refueling problem. P2P strategy is an alternative to the single vehicle refueling system where all satellites share the responsibility of refueling each other on an equal footing.

The remainder of this chapter is organized as follows: In section 2.2, heuristic methods to solve multi-dimensional assignment problems will be provided. Section 2.2.1 describes the method to solve MAPs with large cardinality. In section 2.2.2, the heuristic method for MAPs with large dimensionality is explained. Section 2.3 contains the numerical results and comparison with exact methods, and finally in section **??**, conclusions and future extensions are provided.

2.2 High-quality Solution Sets in Randomized Multidimensional Assignment Problems

In this section two methods will be described that can be used to obtain mathematically proven high-quality solutions for MAPs with large cardinality, or large dimensionality. These methods utilize the concept of *index graph* of the underlying hypergraph of the problem.

2.2.1 Random Linear MAPs of Large Cardinality

In the case when the cost Φ of hypergraph matching is a linear function of hyperedges' costs, i.e., for MAPs with linear objectives, a useful tool for constructing high quality solutions for instances with large cardinality ($n \gg 1$) is the so-called *index graph*. The index graph is related to the concept of *line graph*, in that the vertices of the index graph represent the hyperedges of the hypergraph.

Namely, by indexing each vertex of the index graph $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$ by $(i_1, \ldots, i_d) \in \{1, \ldots, n\}^d$, identically to the corresponding hyperedge of $\mathcal{H}_{d|n}$, the set of vertices $\mathcal{V}^*$ can be partitioned into n subsets $\mathcal{V}_k^*$, also called *levels*, which contain vertices whose first index is equal to k:

$$\mathcal{V}^* = \bigcup_{k=1}^{n} \mathcal{V}_k^*, \quad \mathcal{V}_k^* = \{(k, i_2, \ldots, i_d) \mid i_2, \ldots, i_d \in \{1, \ldots, n\}\}.$$

For any two vertices $i, j \in \mathcal{V}^*$, an edge (i, j) exists in $\mathcal{G}^*$, $(i, j) \in \mathcal{E}^*$, if and only if the corresponding hyperedges of $\mathcal{H}_{d|n}$ do not have common nodes. In other words,

$$\mathcal{E}^* = \{(i, j) \mid i = (i_1, \ldots, i_d), j = (j_1, \ldots, i_d) : \ i_k \neq j_k, \ k = 1, \ldots, n\}.$$

Then, that it is easy to see that $\mathcal{G}^*$ has the following properties.

Lemma 2.2.1. *Consider a complete, d-partite, n-uniform hypergraph $\mathcal{H}_{d|n} = (\mathcal{V}, \mathcal{E})$, where $|\mathcal{E}| = n^d$, and $\mathcal{V} = \bigcup_{k=1}^{d} \mathcal{V}_k$ such that $\mathcal{V}_k \cap \mathcal{V}_l = \emptyset$, $k \neq l$ and $|\mathcal{V}_k| = n$, $k = 1, \ldots, d$. Then, the index graph $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$ of $\mathcal{H}_{d|n}$ satisfies:*

1. *$\mathcal{G}^*$ is n-partite, namely $\mathcal{V}^* = \bigcup_{k=1}^{n} \mathcal{V}_k^*$, $\mathcal{V}_i^* \cap \mathcal{V}_j^* = \emptyset$ for $i \neq j$, where each $\mathcal{V}_k^*$ is an independent set in $\mathcal{V}^*$: for any $i, j \in \mathcal{V}_k^*$ one has $(i, j) \notin \mathcal{E}^*$*
2. *$|\mathcal{V}_k^*| = n^{d-1}$ for each $k = 1, \ldots, n$*

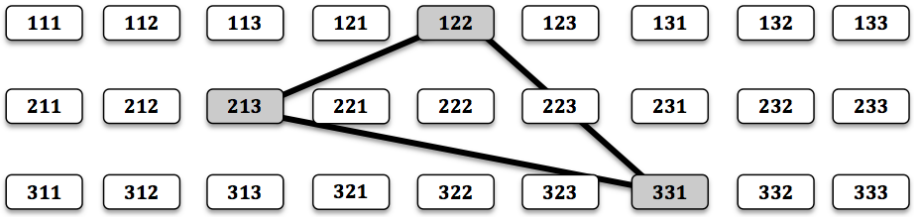


Figure 2.3: The index graph $\mathcal{G}^*$ of the hypergraph $\mathcal{H}_{d|n}$ shown in Figure 2.2. The vertices of $\mathcal{G}^*$ shaded in grey represent a clique (or, equivalently, a perfect matching on $\mathcal{H}_{d|n}$).

3. *The set of perfect matchings in $\mathcal{H}_{d|n}$ is isomorphic to the set of n-cliques in $\mathcal{G}^*$, i.e., each perfect matching in $\mathcal{H}_{d|n}$ corresponds uniquely to a (maximum) clique of size n in $\mathcal{G}^*$.*

Let us denote by $\mathcal{G}^*(\alpha_n)$ the induced subgraph of the index graph $\mathcal{G}^*$ obtained by *randomly* selecting α_n vertices from each level $\mathcal{V}_k^*$ of $\mathcal{G}^*$, and also define $N(\alpha_n)$ to be the number of cliques in $\mathcal{G}^*(\alpha_n)$, then based on the following lemma [38] one can select α_n in such a way that $\mathcal{G}^*(\alpha_n)$ is expected to contain at least one n-clique:

Lemma 2.2.2. *The subgraph $\mathcal{G}^*(\alpha_n)$ is expected to contain at least one n-clique, or a perfect matching on $\mathcal{H}_{d|n}$ (i.e., $E[N(\alpha_n)] \geq 1$) when α_n is equal to*

$$\alpha_n = \left\lceil \frac{n^{d-1}}{n!^{\frac{d-1}{n}}} \right\rceil . \tag{2.10}$$

In the case when the cost coefficients $\phi_{i_1 \cdots i_d}$ of MAP with linear or bottleneck objective are drawn independently from a given probability distribution, Lemma 2.2.2 can be used to construct high quality solutions. The approach is to create the subgraph $\mathcal{G}^*_{\min}(\alpha_n)$, also called the *$\alpha$-set*, from the index graph $\mathcal{G}^*$ of the MAP by selecting

α_n nodes with the smallest cost coefficients from each partition (level) of $\mathcal{G}^*$. If the costs of the hyperedges of $\mathcal{H}_{d|n}$, or, equivalently, vertices of $\mathcal{G}^*$, are identically and independently distributed, the α-set is expected to contain at least one clique, which represents a perfect matching in the hypergraph $\mathcal{H}_{d|n}$. It should be noted that since the α-set is created from the nodes with the smallest cost coefficients, if a clique exists in the α-set, the resulting cost of the perfect matching is expected to be close to the optimal solution of the MAP.

Importantly, when the cardinality n of the MAP increases, the size of the subgraph $\mathcal{G}^*(\alpha_n)$ or $\mathcal{G}^*_{\min}(\alpha_n)$ grows only as $O(n)$, as evidenced by the following observation:

Lemma 2.2.3. *If d is fixed and $n \to \infty$, then α_n monotonically approaches a finite limit:*

$$\alpha_n \nearrow \alpha := \lceil e^{d-1} \rceil \quad as \quad n \nearrow \infty. \tag{2.11}$$

Corollary 2.2.4. *In the case of randomized MAP of large enough cardinality $n \gg 1$ the subset $\mathcal{G}^*_{\min}$ expected to contain a high-quality feasible solution of the MAP can simply be chosen as $\mathcal{G}^*_{\min}(\alpha)$, where α is given by* (2.11).

Observe that using the α-set $\mathcal{G}^*_{\min}(\alpha)$ for construction of a low-cost feasible solution to randomized MAP with linear or bottleneck objectives may prove to be a challenging task, since it is equivalent to finding an n-clique in an n-partite graph; moreover, the graph $\mathcal{G}^*_{\min}(\alpha)$ is only expected to contain a single n-clique (feasible

solution). The following variation of Lemma 2.2.2 allows for constructing a subgraph of $\mathcal{G}^*$ that contains exponentially many feasible solutions:

Lemma 2.2.5. *Consider the index graph $\mathcal{G}^*$ of the underlying hypergraph $\mathcal{H}_{d|n}$ of a randomized MAP, and let*

$$\beta_n = \left\lceil 2\frac{n^{d-1}}{n!^{\frac{d-1}{n}}} \right\rceil. \tag{2.12}$$

Then, the subgraph $\mathcal{G}^*(\beta_n)$ is expected to contain 2^n n-cliques, or, equivalently, perfect matching on $\mathcal{H}_{d|n}$.

Proof. The statement of the lemma is easy to obtain by regarding the feasible solutions of the MAP as *paths* that contain exactly one vertex in each of the n "levels" $\mathcal{V}_1^*, \ldots, \mathcal{V}_n^*$ of the index graph $\mathcal{G}^*$. Namely, let us call a path connecting the vertices $(1, i_2^{(1)}, \ldots, i_d^{(1)}) \in \mathcal{V}_1^*$, $(2, i_2^{(2)}, \ldots, i_d^{(2)}) \in \mathcal{V}_2^*$, $\ldots$, $(n, i_2^{(n)}, \ldots, i_d^{(n)}) \in \mathcal{V}_n^*$ *feasible* if $\{i_k^{(1)}, i_k^{(2)}, \ldots, i_k^{(n)}\}$ is a permutation of the set $\{1, \ldots, n\}$ for every $k = 2, \ldots, d$. Note that from the definition of the index graph $\mathcal{G}^*$ it follows that a path is feasible if and only if the vertices it connects form an n-clique in $\mathcal{G}^*$. Next, observe that a path in $\mathcal{G}^*$ chosen at random is feasible with the probability $\left(\frac{n!}{n^n}\right)^{d-1}$, since one can construct $n^{n(d-1)}$ different (not necessarily feasible) paths in $\mathcal{G}^*$. Then, if we randomly select β_n vertices from each set $\mathcal{V}_k^*$ in such a way that out of the $(\beta_n)^n$ paths spanned by $\mathcal{G}^*(\beta_n)$ at least 2^n are feasible, the value of β_n must satisfy:

$$(\beta_n)^n \left(\frac{n!}{n^n}\right)^{d-1} \geq 2^n,$$

from which it follows immediately that β_n must satisfy (2.12).

Corollary 2.2.6. *If d is fixed and $n \to \infty$, then β_n monotonically approaches a finite limit:*

$$\beta_n \nearrow \beta := \lceil 2e^{d-1} \rceil \quad as \quad n \nearrow \infty.$$

Remark 2.2.7. *Since the value of the parameter β_n (2.12) is close to the double of the parameter α_n (2.10), the subgraph $\mathcal{G}^*_{\min}(\beta_n)$, constructed from selecting β_n nodes with the smallest cost coefficients from each partition (level) of $\mathcal{G}^*$ will be called the "2α-set", or $\mathcal{G}^*(2\alpha)$.*

Following [39], the costs of feasible solutions of randomized MAPs with linear or bottleneck objectives that are contained in the α- or 2α-sets can be shown to satisfy:

Lemma 2.2.8. *Consider a randomized MAP with linear or bottleneck objectives, whose cost coefficients are iid random variables from a continuous distribution F with a finite left endpoint of the support, $F^{-1}(0) > -\infty$. Then, for a fixed $d \geq 3$ and large enough values of n, if the subset $\mathcal{G}^*_{\min}(\alpha)$ (or, respectively, $\mathcal{G}^*_{\min}(\beta)$) contains a feasible solution of the MAP, the cost Z_n of this solution satisfies*

$$(n-1)F^{-1}(0) + F^{-1}\left(\frac{1}{n^{d-1}}\right) \leq Z_n \leq nF^{-1}\left(\frac{3\ln n}{n^{d-1}}\right), \quad n \gg 1, \tag{2.13}$$

in the case of MAP with linear objective (2.7), while in the case of MAP with bottleneck objective (2.8) the cost W_n of such a solution satisfies

$$F^{-1}\left(\frac{1}{n^{d-1}}\right) \leq W_n \leq F^{-1}\left(\frac{3\ln n}{n^{d-1}}\right), \quad n \gg 1. \tag{2.14}$$

2.2.2 Random MAPs of Large Dimensionality

In cases where the cardinality of the MAP is fixed, and its dimensionality is large, $d \gg 1$, the approach described in section 2.2.1 based on the construction of α- or 2α-subset of the index graph $\mathcal{G}^*$ of the MAP is not well suited, since in this case the size of $\mathcal{G}^*(\alpha)$ grows exponentially in d.

However, the index graph $\mathcal{G}^*$ of the underlying hypergraph $\mathcal{H}_{d|n}$ of the MAP can still be utilized to construct high-quality solutions of large-dimensionality randomized MAPs.

Let us call two matchings $\mu_i = \{(i_1^{(1)}, \ldots, i_d^{(1)}), \ldots, (i_1^{(n)}, \ldots, i_d^{(n)})\}$ and $\mu_j = \{(j_1^{(1)}, \ldots, j_d^{(1)}), \ldots, (j_1^{(n)}, \ldots, j_d^{(n)})\}$ on the hypergraph $\mathcal{H}_{d|n}$ *disjoint* if

$$(i_1^{(k)}, \ldots, i_d^{(k)}) \neq (j_1^{(\ell)}, \ldots, j_d^{(\ell)}) \quad \text{for all} \quad 1 \leq k, \ell \leq n,$$

or, in other words, if μ_i and μ_j do not have any common hyperedges. It is easy to see that if the cost coefficients of randomized MAPs are iid random variables, then the costs of the feasible solutions corresponding to the disjoint matchings are also independent and identically distributed.

Next, we show how the index graph $\mathcal{G}^*$ of the MAP can be used to construct exactly n^{d-1} disjoint solutions whose costs are iid random variables. First, recalling the interpretation of feasible MAP solutions as *paths* in the index graph $\mathcal{G}^*$, we observe that disjoint solutions of MAP, or, equivalently, disjoint matchings on $\mathcal{H}_{d|n}$ are represented by disjoint paths in $\mathcal{G}^*$ that do not have common vertices.

Note that since each level $\mathcal{V}_k^*$ of $\mathcal{G}^*$ contains exactly n^{d-1} vertices (see Lemma 2.2.1), there may be no set of disjoint paths with more than n^{d-1} elements.

On the other hand, recall that a (feasible) path $\mathcal{G}^*$ can be described as a set of n vectors

$$\mu = \{(i_1^{(1)}, \ldots, i_d^{(1)}), \ldots, (i_1^{(n)}, \ldots, i_d^{(n)})\},$$

such that $\{i_k^{(1)}, \ldots, i_k^{(n)}\}$ is a permutation of the set $\{1, \ldots, n\}$ for each $k = 1, \ldots, d$. Then, for any given vertex $v^{(1)} = (1, i_2^{(1)}, \ldots, i_d^{(1)}) \in \mathcal{V}_1^*$, let us construct a feasible path containing $v^{(1)}$ in the form

$$\{(1, i_2^{(1)}, \ldots, i_d^{(1)}), (2, i_2^{(2)}, \ldots, i_d^{(2)}), \ldots, (n, i_2^{(n)} \cdots i_d^{(n)})\},$$

where for $k = 2, \ldots, d$ and $r = 2, \ldots, n$

$$i_k^{(r)} = \begin{cases} i_k^{(r-1)} + 1, & \text{if } i_k^{(r-1)} = 1, \ldots, n-1, \\ 1, & \text{if } i_k^{(r-1)} = n. \end{cases} \tag{2.15}$$

In other words, $\{i_k^{(1)}, \ldots, i_k^{(n)}\}$ is a forward cyclic permutation of the set $\{1, \ldots, n\}$ for any $k = 2, \ldots, d$. Applying (2.15) to each of the n^{d-1} vertices $(1, i_2^{(1)}, \ldots, i_d^{(1)}) \in \mathcal{V}_1^*$, we obtain n^{d-1} feasible paths (matchings on $\mathcal{H}_{d|n}$) that are mutually disjoint, since (2.15) defines a bijective mapping between any vertex (hyperedge) $(k, i_2^{(k)}, \ldots, i_d^{(k)})$ from the set $\mathcal{V}_k^*$, $k = 2, \ldots, n$, and the corresponding vertex (hyperedge) $v^{(1)} \in \mathcal{V}_1^*$.

Then, if hyperedge costs $\phi_{i_1 \cdots i_d}$ in the linear or bottleneck MAPs (2.7) and (2.8) are stochastically independent, the costs $\Phi(\mu_1), \ldots, \Phi(\mu_{n^{d-1}})$ of the n^{d-1} disjoint matchings $\mu_1, \ldots, \mu_{n^{d-1}}$ defined by (2.15) are also independent, as they do not contain any common elements $\phi_{i_1 \cdots i_d}$. Given that the optimal solution cost $Z_{d,n}^*$ (respectively, $W_{d,n}^*$) of randomized linear (respectively, bottleneck) MAP does not exceed the costs $\Phi(\mu_1)$,..., $\Phi(\mu_{n^{d-1}})$ of the disjoint solutions described by (2.15), the following bound on the optimal cost of linear or bottleneck randomized MAP can be established:

Lemma 2.2.9. *The optimal costs $Z^*_{d,n}$, $W^*_{d,n}$ of random MAPs with linear or bottleneck objectives* (2.7), (2.8)*, where cost coefficients are iid random variables, satisfy*

$$Z^*_{d,n} \leq X^{\sum}_{1:n^{d-1}}, \quad W^*_{d,n} \leq X^{\max}_{1:n^{d-1}}, \tag{2.16}$$

where $X^{\sum}_i$, $X^{\max}_i$ $(i = 1, \ldots, n^{d-1})$ are iid random variables with distributions $F^{\sum,\max}$ that are determined by the form of the corresponding objective function, and $X_{1:k}$ denotes the minimum-order statistic among k iid random variables.

Remark 2.2.10. *Inequalities in* (2.16) *are tight: namely, in the special case of random MAPs with $n = 2$, all of the $n!^{d-1} = 2^{d-1}$ feasible solutions are stochastically independent [29], whereby equalities hold in* (2.16).

As shown in [39], the following quality guarantee on the minimum cost of the n^{d-1} disjoint solutions (2.15) of linear and bottleneck MAPs can be established:

$$X^{\sum}_{1:n^{d-1}} \leq nF^{-1}\left(n^{-\frac{d-1}{2n}}\right), \quad X^{\max}_{1:n^{d-1}} \leq F^{-1}\left(n^{-\frac{d-1}{2n}}\right), \quad d \gg 1,$$

where F^{-1} is the inverse of the distribution function F of the cost coefficients $\phi_{i_1 \cdots i_d}$. This observation allows for constructing high-quality solutions of randomized linear and bottleneck MAPs by searching the set of disjoint feasible solutions as defined by (2.15).

2.3 Numerical Results

Sections 2.2.1 and 2.2.2 introduced two methods of solving randomized instances of MAPs by constructing subsets (neighborhoods) of the feasible set of the problem that are guaranteed to contain high-quality solutions whose costs approach optimality

when the problem size ($n \to \infty$, or, respectively, $d \to \infty$) increases. In this section we investigate the quality of solutions contained in these neighborhoods for small- to moderate-sized problem instances, and compare the results with the optimal solutions where it is possible.

Before proceeding with the numerical results of the study, in the next section, FINDCLIQUE, the algorithm that is used to find the optimum clique in the index-graph $\mathcal{G}^*$ or the first clique in the α-set or 2α-set will be described. The results from randomly generated MAP instances for each of these two methods are presented next.

2.3.1 Finding n-Cliques in n-Partite Graphs

In order to find cliques in $\mathcal{G}^*$, the α-set, or the 2α-set, the branch-and-bound algorithm proposed in [30] is used. This algorithm, called FINDCLIQUE, is designed to find all n-cliques contained in an unweighted n-partite graph.

The input to original FINDCLIQUE is an n-partite graph $G(V_1, \ldots, V_n; E)$ with the adjacency matrix $\mathsf{M} = (m_{ij})$, and the output will be a list of all n-cliques contained in G. Nodes from G are copied into a set called `compatible nodes`, denoted by C. The set C is further divided into n partitions, each denoted by C_i that are initialized such that they contain nodes from partite V_i, $i = \{1, \ldots, n\}$. FINDCLIQUE also maintains two other sets, namely, `current clique`, denoted by Q and `erased nodes`, denoted by E. The set Q holds a set of nodes that are pairwise adjacent and construct a clique. The `erased node` set, E, is furthered partitioned into n sets, denoted by

E_i, that are initialized as empty. At each step of the algorithm, E_i will contain the nodes that are not adjacent to the i^{th} node added to Q.

The branch-and-bound tree has n levels, and FINDCLIQUE searches for n-cliques in the tree in a depth-first fashion. At level t of the branch of bound algorithm, the index of the smallest partition in C, $\theta = \arg\min_i\{|C_i| \,\big|\, i \notin \mathsf{V}\}$ will be detected, and C_θ will be marked as visited by including θ into $\mathsf{V} \leftarrow \{\mathsf{V} \cup \theta\}$, where V is the list of partitions that have a node in Q. Then, a node q from C_θ is selected at random and added to Q. If $|Q| = n$, an n-clique is found. Otherwise, C will be updated; every partition C_i where $i \notin \mathsf{V}$ will be searched for nodes c_{ij}, $(j = 1, \ldots, |C_i|)$ that are not adjacent to q, i.e. $m_{q,c_{ij}} = 0$. Any such node will be removed from C_i and will be transferred to E_t. Note that in contrast to C, nodes in different levels of E will not necessarily be from the same partite of G. Decision regarding backtracking is made after C is updated. It is obvious that in an n-partite graph the following will hold:

$$\omega(G) \leq n, \tag{2.17}$$

where $\omega(G)$ is the size of a maximum clique in G. In other words, the size of any maximum clique cannot be larger than the number of partites, in that the maximum clique can only contain at most 1 node from each partite of G. If after updating, there is any $C_i \notin \mathsf{V}$ with $|C_i| = 0$, adding q_i to Q will not result in a clique of size n, since the condition in (2.17) changes into strict inequality. In such cases, q is removed from Q, nodes from E_t will be transferred back to their respective partitions in C, and FINDCLIQUE will try to add another node from C_θ that is not already

branched on, to Q. If such a node does not exist, the list of visited partitions will be updated ($\mathsf{V} \leftarrow \mathsf{V} \backslash \theta$), and FINDCLIQUE backtracks to the previous level of the branch-and-bound tree. If the backtracking condition is not met and q is promising, FINDCLIQUE will go one level deeper in the tree, finds the next smallest partition in the updated C and tries to add a new node to Q.

When solving the clique problem in the α-set or 2α-set, since the objective is to find the first n-clique regardless of its cost, FINDCLIQUE can be used without any modifications, and the weights of the nodes in $\mathcal{G}^*_{\min}(\alpha)$ or $\mathcal{G}^*_{\min}(2\alpha)$ will be ignored. However, when the optimal clique with the smallest cost in $\mathcal{G}^*$ is sought, some modifications in FINDCLIQUE are necessary to enable it to deal with weighted graphs. The simplest way to adjust FINDCLIQUE is to compute the weight of the n-cliques as they are found, and report the clique with the smallest cost as the output of the algorithm. This is the method that is used in the experimental studies whenever the optimal solution is desired. However, to obtain a more efficient algorithm, it is possible to calculate the weight of the partial clique contained in Q in every step of the algorithm and fathom subproblems for which $W_Q \geq W_{Q^*}$, where W_Q and W_{Q^*} are the cost of the partial clique in Q and the cost of the best clique found so far by the algorithm respectively. Further improvement can be achieved by sorting the nodes in C_i, $i = 1, \ldots, n$, based on their cost coefficients, and each time select the untraversed node with the smallest node as the next node to be added to Q (as opposed to randomly selecting a node, which does not change the overall computational time in the unweighted graph if a list of all n-cliques is desired). This enables us to compute a

lower bound on the cost of the maximum clique that the nodes in Q may lead to as follows:

$$LB_Q = W_Q + \sum_{i \notin \mathsf{V}} w_i^{\min}, \tag{2.18}$$

where $w_i^{\min}$ is the weight of the node with the smallest cost coefficient in C_i. Any subproblem with $LB_Q \geq W_{Q^*}$ will be fathomed.

2.3.2 Random Linear MAPs of Large Cardinality

To demonstrate the performance of the method described in section 2.2.1, random MAPs with fixed dimensionality $d = 3$ and different values of cardinality n are generated. The cost coefficients $\phi_{i_1 \cdots i_d}$ are randomly drawn from the uniform $U[0, 1]$ distribution. Three sets of problems are solved for this case: (i) $n = 3, \ldots, 8$ with $d = 3$, solved for optimality, and the first clique in the α- and 2α-sets, (ii) $n = 10, 15, \ldots, 45$, with $d = 3$, solved for the first clique in the α- and 2α-sets, and finally (iii) $n = 50, 55, \ldots, 80$, with $d = 3$, solved for the first clique in the 2α-set. For each value of n, 25 instances are generated and solved by modified FINDCLIQUE for the optimum clique or FINDCLIQUE whenever the first clique in the problem is desired. Algorithm is terminated if the computational time needed to solve an instance exceeds 1 hour.

In the first group, (i), instances of MAP that admit solution to optimality in a reasonable time were solved. The results from this subset are used to determine the applicability of Corollary 2.2.4 and bounds (2.13) and (2.14) for relatively small values of n. Table 2.1 summarizes the average values for the cost of the clique and

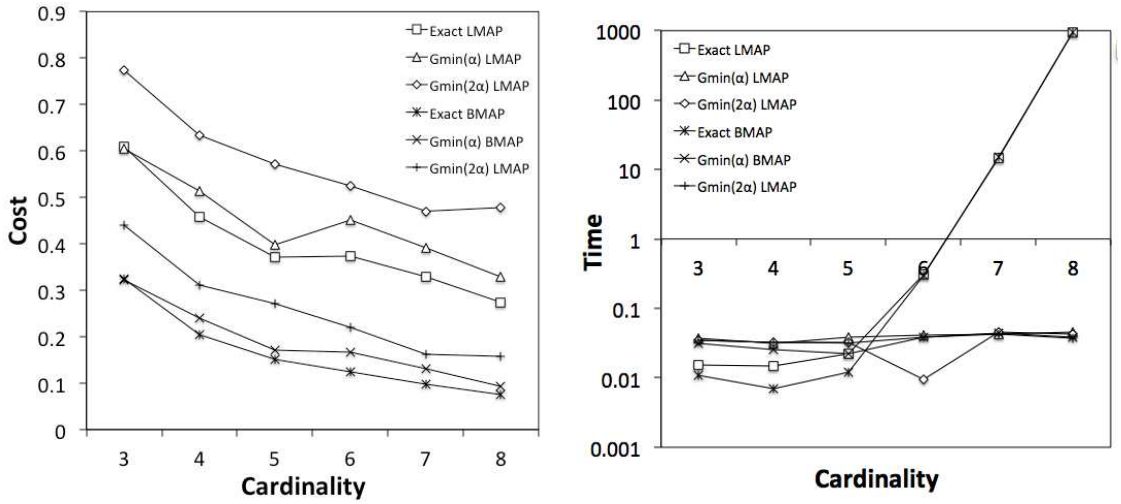
computational time needed for MAPs with the linear sum objective function for the instances in group (i). The first column, n, is the cardinality of the problem. The columns under the heading "Exact" contain the values related to the optimal clique in $\mathcal{G}^*$. The columns under the heading "$\mathcal{G}^*_{\min}(\alpha_n)$" represent the values obtained from solving the α-set for the first clique, and those under the heading "$\mathcal{G}^*_{\min}(2\alpha)$" represent the values obtained from solving the 2α-set for the first clique. For each of these multicolumns, T denotes the average computational time in seconds, Z is the average cost of the cliques, $|V|$ is the order of the graph or induced subgraph in $\mathcal{G}^*$, $\mathcal{G}^*_{\min}(\alpha)$, or $\mathcal{G}^*_{\min}(2\alpha)$, and $\exists$ CLQ shows the percentage of the problems for which the α-set or 2α-set, respectively, contains a clique. This value is 100% for the exact method. There was no instances in group (i) for which the computational time exceeded 1 hour.

It is clear that using α-set or 2α-set enables us to obtain a high-quality solution in a much shorter time by merely searching a significantly smaller part of the index graph $\mathcal{G}^*$. Based on the values for Z, the cost of the clique found in α-set or 2α-set are consistently converging to that of the optimal clique and they provide tight upper bounds for the optimum cost. Additionally, as is shown in the $|V|$ column, significant reduction in the size of the graph can be obtained if α-set or 2α-set are used.

Table 2.2 contains the corresponding results for the case of a random MAP with bottleneck objective. In this table, W represents the value for the cost of the optimal clique or the first clique found in α- or 2α-set. Figure 2.4(a) shows how the cost of an optimum clique compares to the cost of the clique found in α-set and 2α-set. Clearly,

the cost of optimal clique approaches 0 for both linear sum and linear bottleneck MAPs. Figure 2.4(b) demonstrates the computational time for instances in group (i).

The advantage of using α-set over 2α-set is that the quality of the detected clique is expected to be higher. On average, however, a clique in 2α-set is found in a shorter time than in α-set.



(a) (b)

Figure 2.4: Behaviour of the solutions obtained from the heuristics: solution costs **(a)** and computational time **(b)** in random MAPs with linear sum and linear bottleneck objective functions for instances in group (i).

Table 2.1: Comparison of the computational time and cost for the optimum clique and the first clique found in $\mathcal{G}^*(\alpha)$ and $\mathcal{G}^*(2\alpha)$ in random MAPs with linear sum objective functions for instances in group (i).

	Exact				$\mathcal{G}^*_{\min}(\alpha)$				$\mathcal{G}^*_{\min}(2\alpha)$			
n	$T^*_{n,3}$	$Z^*_{n,3}$	$\|V\|$	$\exists$ CLQ	$T_{\mathcal{G}_{min}(\alpha)}$	$Z_{\mathcal{G}_{min}(\alpha)}$	$\|V\|$	$\exists$ CLQ	$T_{\mathcal{G}_{min}(2\alpha)}$	$Z_{\mathcal{G}_{min}(2\alpha)}$	$\|V\|$	$\exists$ CLQ
3	0.02	0.604	3×26	100	0.04	0.609	3×3	76	0.03	0.773	3×6	100
4	0.01	0.458	4×63	100	0.03	0.514	4×4	88	0.03	0.635	4×7	100
5	0.02	0.371	5×124	100	0.04	0.399	5×4	72	0.03	0.571	5×8	100
6	0.31	0.374	6×215	100	0.04	0.452	6×5	92	0.01	0.524	6×9	100
7	14.83	0.329	7×342	100	0.04	0.392	7×5	80	0.05	0.47	7×9	100
8	937.67	0.274	8×511	100	0.05	0.329	8×5	72	0.04	0.478	8×10	100

Table 2.2: Comparison of the computational time and cost for the optimum clique and the first clique found in $\mathcal{G}^*(\alpha)$ and $\mathcal{G}^*(2\alpha)$ in random MAPs with linear bottleneck objective functions for instances in group (i).

	Exact				$\mathcal{G}^*_{\min}(\alpha)$				$\mathcal{G}^*_{\min}(2\alpha)$			
n	$T^*_{n,3}$	$W^*_{n,3}$	$\|V\|$	$\exists$ CLQ	$T_{\mathcal{G}_{min}(\alpha)}$	$W_{\mathcal{G}_{min}(\alpha)}$	$\|V\|$	$\exists$ CLQ	$T_{\mathcal{G}_{min}(2\alpha)}$	$W_{\mathcal{G}_{min}(2\alpha)}$	$\|V\|$	$\exists$ CLQ
3	0.01	0.321	3×26	100	0.03	0.324	3×3	76	0.04	0.439	3×6	100
4	0.01	0.205	4×63	100	0.03	0.241	4×4	88	0.03	0.311	4×7	100
5	0.01	0.151	5×124	100	0.02	0.17	5×4	72	0.03	0.27	5×8	100
6	0.3	0.124	6×215	100	0.04	0.166	6×5	92	0.04	0.219	6×9	100
7	14.96	0.098	7×342	100	0.04	0.131	7×5	80	0.04	0.163	7×9	100
8	956.6	0.075	8×511	100	0.04	0.092	8×5	72	0.04	0.157	8×10	100

The second group of problems, (ii), comprises instances that cannot be solved to optimality within 1 hour. The range of n for this group is such that the first clique in the α-set is expected to be found within 1 hour. Tables 2.3 and 2.4 summarize the results obtained for this group. Instances with $n = 45$ were the largest problems in this group for which α-set could be solved within 1 hour. As it is expected, the 2α-set can be solved quickly in a matter of seconds where the equivalent problem for α-set requires a significantly longer computational time. However, the quality of the solutions found for α-set is higher than the quality for solutions in 2α-set. Nonetheless, using 2α-set increases the odds of finding a clique, as based on lemma 2.2.5, 2α-set is expected to contain an exponential number of cliques. It is obvious from the $\exists$ CLQ column that not all of the instances in α-set contain at least a clique, whereas 100% of the instances in 2α-set contain one that can be found within 1 hour. Column *Timeout* represents the percentage of the problems that could not be solved within the allocated 1 hour time limit. Out of 25 instances solved for $n = 45$, only 4 (16%) could not be solved in 1 hour. Out of the 21 remaining instances, 20 instances contained a clique, and only 1 did not have a clique. The behavior of the average cost values for the problems solved in this group are depicted in Figure 2.5.

Table 2.3: Comparison of the computational time and cost for the first clique found in $\mathcal{G}^*(\alpha)$ and $\mathcal{G}^*(2\alpha)$ in random MAPs with linear sum objective functions for instances in group (ii).

	$\mathcal{G}^*_{\min}(\alpha)$					$\mathcal{G}^*_{\min}(2\alpha)$				
n	$T_{\mathcal{G}_{min}(\alpha)}$	$Z_{\mathcal{G}_{min}(\alpha)}$	$\lvert V\rvert$	$\exists$ CLQ	Timeout	$T_{\mathcal{G}_{min}(2\alpha)}$	$Z_{\mathcal{G}_{min}(2\alpha)}$	$\lvert V\rvert$	$\exists$ CLQ	Timeout
10	0.05	0.266	10×5	60	-	0.05	0.37	10×10	100	-
15	0.06	0.228	15×6	76	-	0.06	0.313	15×11	100	-
20	0.08	0.165	20×6	56	-	0.07	0.246	20×12	100	-
25	0.15	0.147	25×7	80	-	0.08	0.2	25×13	100	-
30	0.89	0.134	30×7	92	-	0.09	0.171	30×13	100	-
35	8.54	0.11	35×7	88	-	0.14	0.151	35×13	100	-
40	100.85	0.097	40×7	92	-	0.46	0.131	40×13	100	-
45	405.16	0.085	45×7	80	16	1.09	0.122	45×14	100	-

Table 2.4: Comparison of the computational time and cost for the first clique found in $\mathcal{G}^*(\alpha)$ and $\mathcal{G}^*(2\alpha)$ in random MAPs with linear bottleneck objective functions for instances in group (ii).

	$\mathcal{G}^*_{\min}(\alpha)$					$\mathcal{G}^*_{\min}(2\alpha)$				
n	$T_{\mathcal{G}_{min}(\alpha)}$	$W_{\mathcal{G}_{min}(\alpha)}$	$\lvert V\rvert$	$\exists$ CLQ	Timeout	$T_{\mathcal{G}_{min}(2\alpha)}$	$W_{\mathcal{G}_{min}(2\alpha)}$	$\lvert V\rvert$	$\exists$ CLQ	Timeout
10	0.04	0.065	10×5	60	-	0.02	0.098	10×10	100	-
15	0.04	0.037	15×6	76	-	0.02	0.056	15×11	100	-
20	0.05	0.023	20×6	56	-	0.04	0.036	20×12	100	-
25	0.1	0.017	25×7	80	-	0.08	0.025	25×13	100	-
30	0.87	0.012	30×7	92	-	0.1	0.019	30×13	100	-
35	8.53	0.009	35×7	88	-	0.15	0.015	35×13	100	-
40	100.99	0.007	40×7	92	-	0.46	0.011	40×13	100	-
45	403.52	0.006	45×7	80	16	1.09	0.009	45×14	100	-

Table 2.5: Computational time and cost for the first clique found in $\mathcal{G}^*(2\alpha)$ in random MAPs with linear sum objective functions for instances in group (iii).

	$\mathcal{G}^*_{\min}(2\alpha)$				
n	$T_{\mathcal{G}_{min}(2\alpha)}$	$Z_{\mathcal{G}_{min}(2\alpha)}$	$\lvert V\rvert$	$\exists$ CLQ	Timeout
50	1.56	0.11	50×14	100	-
55	52.29	0.099	55×14	96	4
60	189.9	0.091	60×14	92	8
65	568.9	0.085	65×14	96	4
70	919.79	0.078	70×14	64	36
75	1556.89	0.075	75×14	40	60
80	1641.26	0.07	80×14	16	84

Finally, the third group, (iii), includes instances for which the cardinality of the problem prevents the α-set from being solved within 1 hour. Thus, for this set, only the 2α-set is used. The instances of this group were solved with the parameter values $n = 50, 55, \ldots, 80$ and $d = 3$. Tables 2.5 and 2.6 summarize the corresponding results. When the size of the problem $n \geq 55$, some instances of problems become impossible to solve within 1 hour time limit. The average cost for the instances that are solved keeps the usual trend and converges to 0 as n grows. The largest problems attempted to be solved in this group are MAPs with $n = 80$. Out of 25 instances of this size, only 4 could be solved within 1 hour. Figure 2.5(a) the average values of solution cost and computational time for the instances of both linear sum and linear bottleneck MAPs. Note that as the size of the problem increases, the reduction in the size of problem achieved from using α-set or 2α-set becomes significantly larger. For instance, in MAP with $n = 80$ and $d = 3$, the 2α-set has 80×14 nodes, while the complete index graph will have 80×80^2 nodes.

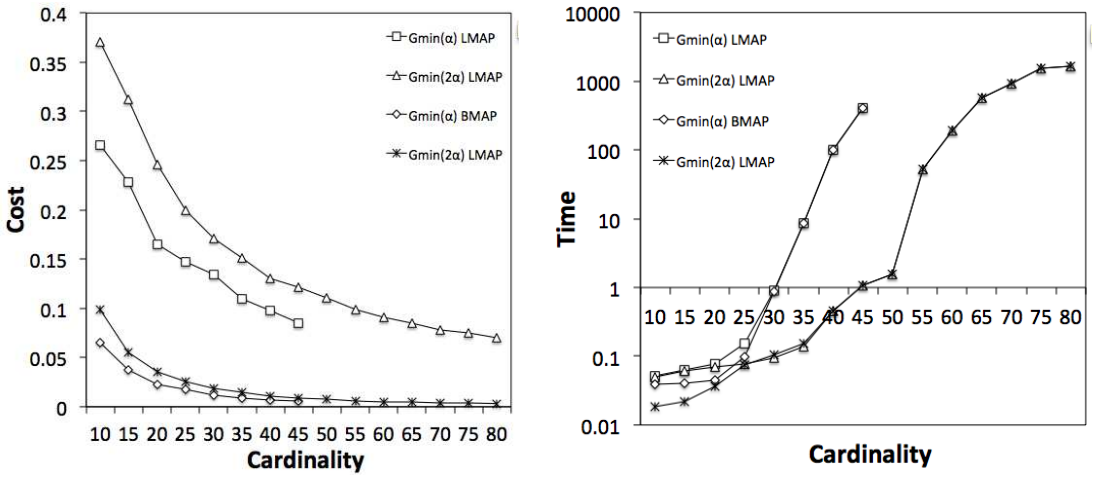
Table 2.6: Computational time and cost for the first clique found in $\mathcal{G}^*(2\alpha)$ in random MAPs with linear bottleneck objective functions for instances in group (iii).

	$\mathcal{G}^*_{\min}(2\alpha)$				
n	$T_{\mathcal{G}_{min}(2\alpha)}$	$W_{\mathcal{G}_{min}(2\alpha)}$	$\|V\|$	$\exists$ CLQ	Timeout
50	1.56	0.008	50×14	100	-
55	52.19	0.006	55×14	96	4
60	190.6	0.005	60×14	92	8
65	566.71	0.005	65×14	96	4
70	920.44	0.004	70×14	64	36
75	1552.74	0.004	75×14	40	60
80	1631.89	0.003	80×14	16	84

2.3.3 Random MAPs of Large Dimensionality

The second set of problem instances includes MAPs that are solved by the heuristic method explained in section 2.2.2. Problems in this set have the cardinality $n = 2, \ldots, 5$ and dimensionality in the range $d = 2, \ldots, \bar{d}_n$, where $\bar{d}_n$ is the largest value for d for which an MAP with cardinality n can be solved within 1 hour using the heuristic method. For each pair of (n, d), 25 instances of MAP with cost coefficients randomly drawn from the uniform $U[0, 1]$ distribution are generated. Generated instances are then solved by the modified FINDCLIQUE for the optimal clique (when possible) and the optimal costs are compared with the costs obtained from the heuristic method. The result of the heuristic method for instances with $n = 2$ is optimal, and the heuristic checks all the 2^{d-1} solutions of the MAP. Thus, using the modified FINDCLIQUE to find the optimum clique is not necessary.

Figure 2.6 demonstrates the cost convergence in instances with $n = 2, 3, 4, 5$ for both linear sum and linear bottleneck MAPs. Figure 2.6(a) demonstrates the cost convergence in MAPs with $n = 2$ and $d = 2, \ldots, 27$. Recall that due to Remark 2, for cases with $n = 2$ the heuristic provides the optimal solution. The heuristic method



(a) (b)

Figure 2.5: Bahaviour of the solution from the heuristic: comparison of the cost **(a)** and computational time **(b)** for MAPs with linear sum and linear bottleneck objective functions for group (ii) and (iii).

provides high quality solutions that are consistently converging to the optimal solution for all cases and the average value of the obtained costs from the heuristics approaches 0. Memory limitations, as opposed to computational time, were the restrictive factor for solving larger instances as the computational time for the problems of this set never exceeded 700 seconds.

Figure 2.7 demonstrates the computational time for the optimal method as well as the heuristic method in instances with $n = 2, 3, 4, 5$ for both linear sum and linear bottleneck MAPs. The computational time has an exponential trend as the number of solutions for the MAP, or the number of solutions checked by the heuristic grow in an exponential manner.

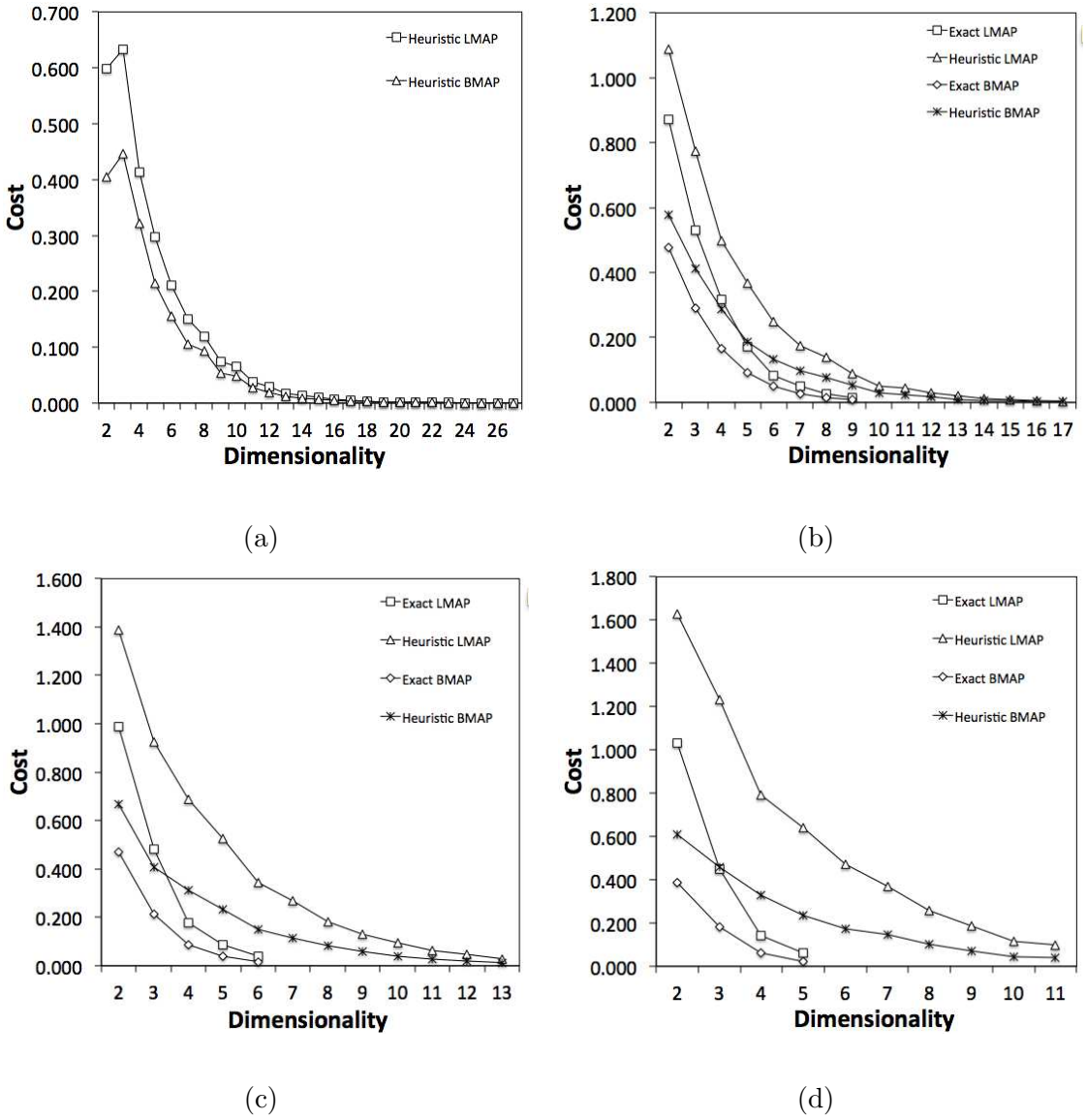


Figure 2.6: Comparison of the cost obtained from the heuristic method with the optimum cost in MAPs with linear sum and linear bottleneck objective functions with **(a)** $n = 2$, **(b)** $n = 3$, **(c)** $n = 4$, and **(d)** $n = 5$

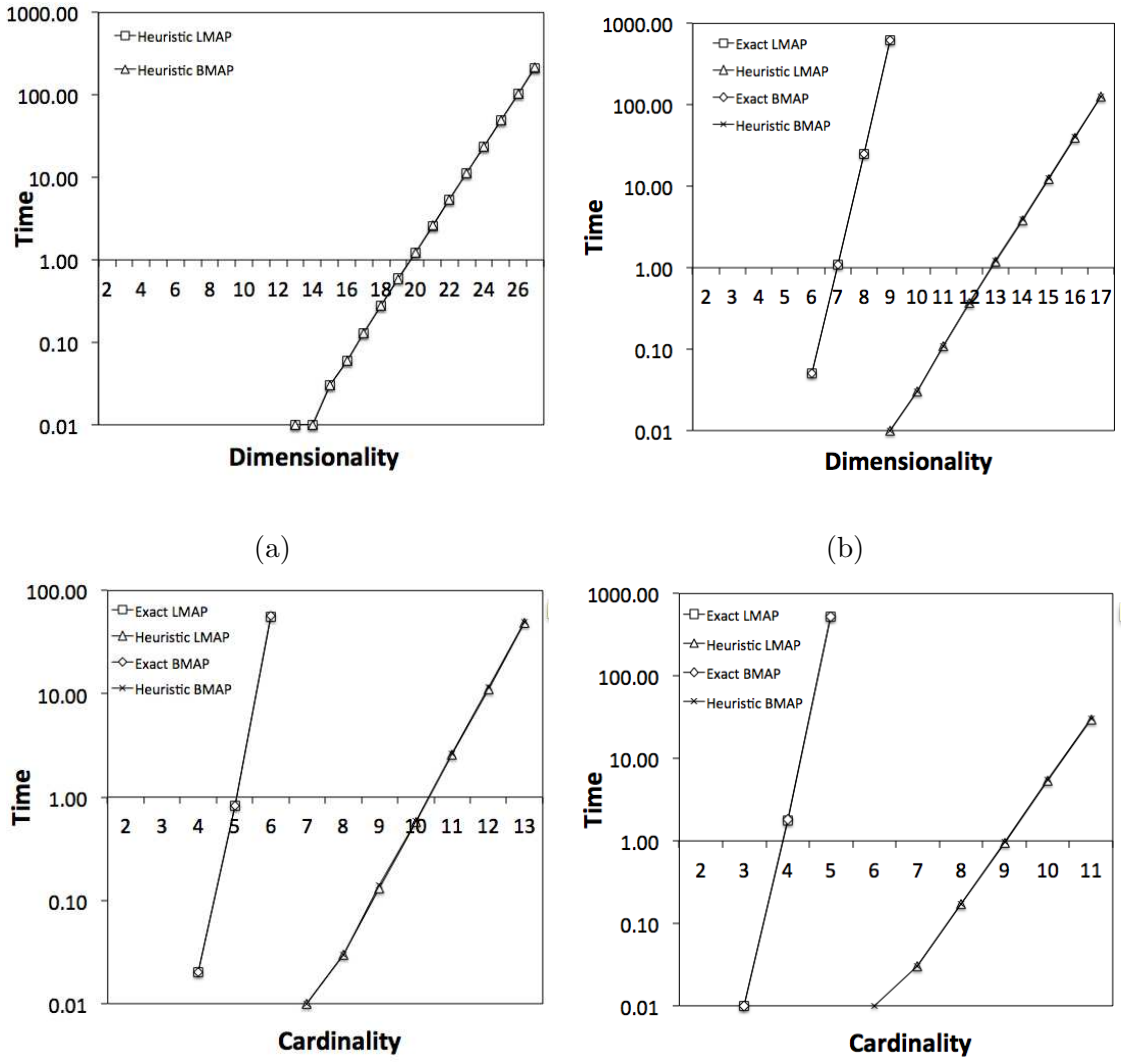


Figure 2.7: Comparison of the computational time in logarithmic scale needed for the optimal method and the heuristic method in MAPs with linear sum and linear bottleneck objective functions with **(a)** $n = 2$, **(b)** $n = 3$, **(c)** $n = 4$, and **(d)** $n = 5$

CHAPTER 3
ON FINDING K-CLIQUES IN K-PARTITE GRAPHS

3.1 Introduction

Given an (undirected) graph $G = (V, E)$, where V is set of nodes and E is the set of arcs, a *clique* in G is defined as a complete subset of G, i.e., a set of nodes in V that are pairwise adjacent. A clique of size k is called k-*clique*; the largest clique in a graph is called the *maximum clique* and its size is denoted by $\omega(G)$. Note that G may contain several cliques of size $\omega(G)$. Closely related to the concept of a clique is that of an *independent set* of G, defined as an induced subgraph of V whose nodes are pairwise disjoint.

The Maximum Clique Problem (MCP) consists in finding the largest clique in a graph, and is of fundamental importance in discrete mathematics, computer science, operations research, and related fields [10]. In many applications it is of interest to identify all maximum cliques in a graph. This problem is known as the Maximum Clique Enumeration Problem (MCEP). In the present work, we consider a special case of the MCEP, concerned with finding all k-cliques in a k-partite graph. A graph $G = (V, E)$ is called k-partite if the set of nodes V can be partitioned into k independent sets, or *partites* V_r, $r = 1, \ldots, k$:

$$V = \bigcup_{r=1}^{k} V_r, \quad V_r \cap V_s = \emptyset, \quad r \neq s, \text{ such that for all } i, j \in V_r : (i, j) \notin E. \tag{3.1}$$

Clearly, one has that $\omega(G) \leq k$ in a k-partite graph G, since the maximum clique cannot contain more than one node from each independent set V_r. The problem of

finding k-cliques in k-partite graphs has found applications in textile industry [30], data mining and clustering [56], and identification of protein structures [47]. This problem is not necessarily equivalent to MCEP since it does not account for maximum cliques with $\omega(G) < k$.

Grunert et al [30] proposed branch-and-bound algorithm FINDCLIQUE for the problem of finding all k-cliques in k-partite graphs, which takes as an input a graph $G = (V, E)$, where V satisfies (3.1), and produces the set Q of k-cliques contained in G as an output. FINDCLIQUE is a recursive method, such that level t of recursion corresponds to the level t of branch-and-bound tree, which in turn, is associated with the t-th partite that is branched on in V. Starting at the root ($t = 0$) of the branch-and-bound tree with a partial solution $S = \emptyset$, at each step of branch-and-bound procedure a node is added to or removed from S until S amounts to a k-clique in G, i.e., $|S| = k$, or it is verified that G contains no k-cliques, $\omega(G) < k$.

Let $B = \{1, \ldots, k\}$ be the index set of partites in G, $V = \bigcup_{b \in B} V_b$, and B_S denote the set of partites that have a node in S:

$$B_S = \{b \in B \mid V_b \cap S \neq \emptyset\}.$$

Given a partial solution S, a node is called *compatible* if it is adjacent to all the nodes in S; the set of compatible nodes w.r.t. S is denoted by C_S:

$$C_S = \{i \in V \mid (i, j) \in E \ \ \forall j \in S\}.$$

The set C_S is further partitioned into subsets containing nodes from the same par-

tite:

$$C_S = \bigcup_{b \in \overline{B}_S} C_{S,b},$$

where $\overline{B}_S = B \setminus B_S$, and $C_{S,b} \subseteq V_b$ is given by

$$C_{S,b} = \bigcup_{s \in S} (V_b \cap N(s)),$$

with $N(s)$ being the set of nodes adjacent to node s.

At the root node of the branch-and-bound tree ($t = 0$), one has $S = \emptyset$, $B = \overline{B}_S = \{1, \ldots, k\}$, $B_S = \emptyset$, and $C_{S,b} = V_b$ for all $b \in B$. At a level t of the branch-and-bound tree, $b_t \in \overline{B}_S$ is selected as the partition to branch on. In order to achieve the greatest reduction in the size of the branch-and-bound tree when pruning, b_t is selected as the partition with the smallest number of nodes:

$$b_t \in \arg\min_b \{|C_{S,b}| \mid b \in \overline{B}_S\}. \tag{3.2}$$

As long as there is a node $n_t \in C_{S,b_t}$ that is not traversed, the search process is restarted from this point with $S := S \cup \{n_t\}$ as the new partial solution. To this end, the set C_S of compatible nodes is updated with respect to $S \cup \{n_t\}$:

$$C_{S,b} := C_{S,b} \cap N(n_t) \quad \text{for all} \quad b \in \overline{B}_S. \tag{3.3}$$

Maintaining the sets $C_{S,b}$ of nodes compatible with the current partial solution S is a key aspect of the algorithm, thus for backtracking purposes the nodes that are removed from $C_{S,b}$ during (3.3) are added to the set $\overline{C} = \bigcup_{t=1}^{k} \overline{C}_t$, which is similarly partitioned into k levels $\overline{C}_t$, each level corresponding to level t of the branch-and-bound tree. In other words, $\overline{C}_t$ contains the nodes in $C_{S,b}$ that are not adjacent to

node n_t:

$$\overline{C}_t = \{i \in C_{S,b} \mid (i, n_t) \notin E,\ b \in \overline{B}_S\}.$$

Obviously, after this step, $C_{S,b_t} = \emptyset$. A subproblem with a partial solution S is *promising* if all of the partitions in C_S that do not share a node in the partial solution are nonempty:

$$|C_{S,b}| > 0 \ \text{ for all } \ b \in \overline{B}_S, \ \ b \neq b_t. \tag{3.4}$$

Let P be the number of partitions $C_{S,b} \subseteq C_S$ that contain at least one node; then, an upper bound on the size of the largest clique containing S is given by $|S| + P$. If $|S| + P = k$, the current subproblem is feasible, meaning S may be part of a k-clique. For a feasible subproblem, the algorithm traverses deeper into the branch-and-bound tree, $t := t + 1$, and a new subproblem is created.

Accordingly, a subproblem with partial solution S is pruned if

$$|S| + P < k, \tag{3.5}$$

i.e., there exists no clique of size k that contains S. For a nonpromising subproblem, set C_{S,b_t} is restored by moving the nodes in $\overline{C}_t$ back to C_S, $C_S := C_S \cup \overline{C}_t$. The last operation implicitly requires that the nodes from $\overline{C}_t$ are put back into the partitions of C_S that they were removed from:

$$C_{S,\pi(v)} := C_{S,\pi(v)} \cup v \ \text{ for all } \ v \in \overline{C}_t, \tag{3.6}$$

where $\pi(i)$ is the index of the partite that node i belongs to: $i \in V_{\pi(i)}$; moreover, the relative orders of nodes in the partites V_b should be preserved in $C_{S,b}$, given that the nodes in G are assumed to be ordered/numbered.

The search process is then restarted, provided that there exists a node in partition C_{S,b_t} that is not traversed. If there is no such node, FINDCLIQUE returns to the previous level $t-1$ of the branch-and-bound tree.

3.2 A bitwise algorithm for finding k-cliques in a k-partite graph

In this section, we present an algorithm, referred to as BitCLQ, for the k-clique enumeration problem in a k-partite graph, which improves upon the FINDCLIQUE algorithm of Grunert et al [30] by introducing bitset data structures and utilizing bit parallelism for updating the set of compatible nodes and improving backtracking.

3.2.1 Bitsets

Bitsets are essentially binary vectors, or sequences of bits, and as such can be utilized efficiently in computer codes. Particularly, bitsets are useful for storing adjacency matrices of graphs, or specific subsets of ordered sets. For example, in a graph on six nodes $\{v_1, \ldots, v_6\} = V$, a clique with nodes v_1, v_2, v_3, v_5 can be represented by a bitset $\{111010\}$, where each bit corresponds uniquely to a node in the graph, with the *significant* bits (i.e., bits equal to 1) indicating the nodes in the clique. *Bit parallelism* is a form of parallel computing that achieves computational improvements by representing the problem data in bitsets of size R, where R is the machine word size (e.g., 32 or 64), such that they can be processed together within a single processor instruction. Bit parallelism has been successfully used in many computational algorithms, particularly for string matching [27, 31, 32]. Recently, bit parallelism has

been employed for solving hard combinatorial problems, such as SAT [66] and the Maximum Clique Problem [65].

In the present work, bit parallelism is used to improve the computational procedure for updating the set of compatible nodes in (3.3), and, moreover, to achieve faster backtracking by eliminating the need for set $\overline{C}$. In addition, use of bitsets allows for improvements in memory storage efficiency for problem data structures, such as the set of compatible nodes and the adjacency matrix of the graph.

Of particular significance in the context of the present work is the operation of indexing the first significant bit in a bitset, also known as the forward bit scanning. One of the techniques for this purpose relies on use of the De Bruijn sequence with a perfect hash table [45]. The value to be looked up in the hash table is given by H_R below:

$$H_R := (x \wedge -x)\, D \gg (R - \log_2 R), \tag{3.7}$$

where x is the bitset for which the first significant bit has to be indexed, D is an instance of De Bruijn sequence, R is the machine word size, and $\gg$ stands for the binary *shift right* operator. H_R is effective for bitsets of maximum size equal to R. For larger bitsets, special containers need to be devised. The hash table required to look up the value of H_R is created based on the particular De Bruijn sequence used in (3.7).

Note that in (3.7) multiplication is performed modulo R and only the last $\log_2 R$ bits of the result will be retained. More details on forward bit scanning and the specification of the De Bruijn sequence used in (3.7) can be found in [45].

3.2.2 BitCLQ

Below we present a modification of FINDCLIQUE, which we refer to as BitCLQ, that uses bitset data structures and bit parallelism for keeping track of the nodes in G that are compatible to the current partial solution S, while simultaneously reducing the computational cost of backtracking.

To this end, we introduce a set Z consisting of k levels, $Z_1, \ldots, Z_k$. Each of these k levels will be used to represent the compatible nodes to the partial solution S at the t-th level of the branch-and-bound tree, where $1 \leq t \leq k$. Every level in Z is further partitioned into k sets, each corresponding to a partite V_b in G:

$$Z_t = \bigcup_{b \in B} Z_{t,b}, \quad t = 1, \ldots, k.$$

The sets $Z_{t,b}$ are represented by bitsets of size $|V_b|$. Let $Z_{t,b,i}$ be the i-th bit in $Z_{t,b}$ corresponding to the i-th node in V_b, such that $Z_{t,b,i} = 1$ if the i-th node in V_b is compatible with all the nodes in the partial solution S at the t-th level of the branch-and-bound tree in BitCLQ:

$$Z_{t,b,i} = \begin{cases} 1, & \text{if } (i,j) \in E \text{ for all } j \in S_t; \\ 0, & \text{otherwise.} \end{cases}$$

Clearly, each level Z_t of Z is an ordered set of combination of bitsets with the total size $|V|$. Further, the adjacency matrix M of graph G is stored in the bitset form, with the convention that the i-th row (column) corresponds to the i-th bit in Z_t, $t = 1, \ldots, k$.

BitCLQ is initialized by setting $t := 0$, $S := \emptyset$, $B = \overline{B}_S := \{1, \ldots, k\}$, and $Q := \emptyset$, where Q is the set of all k-cliques in G. Note that since at the beginning all the

nodes in G can be added to S to extend its size, all the bits in Z_1 are significant:

$$Z_{1,b,i} = 1 \text{ for all } b \in \overline{B}(S_t),\ i \in V_b.$$

At level t of the branch-and-bound tree, the partition b_t to branch on is selected as

$$b_t \in \arg\min_b \{|Z_{t,b}| \mid b \in \overline{B}_S\}, \tag{3.8}$$

where $|Z_{t,b}|$ is defined as the number of significant bits in the bitset $Z_{t,b}$. The forward bit scanning method discussed in Section 3.2.1 is used to identify node $n_t \in V_{b_t}$ that has not been traversed and thus can be added to the partial solution. As long as such a node exists in V_{b_t}, the search process is restarted with $S := S \cup \{n_t\}$ as the partial solution, and the corresponding bit in Z_{t,b_t} is set to 0.

Utilizing bitsets also facilitates the process of updating the compatible nodes: when n_t is added to partial solution, Z_{t+1} is created by performing a logical AND operation with Z_t and the row $M(n_t)$ of the adjacency matrix corresponding to the node n_t as operands:

$$Z_{t+1} = Z_t \wedge M(n_t). \tag{3.9}$$

Similarly to FINDCLIQUE, let P denote the number of partitions $Z_{t,b}$ with $|Z_{t,b}| > 0$ at level the t of the branch-and-bound tree. If $|S| + P = k$, the current partial solution is promising, so that a new subproblem is created, and BitCLQ proceeds one level deeper into the branch-and-bound tree, $t := t + 1$. If the partial solution is not promising, the method presented in Section 3.2.1 is used to select nodes in V_{b_t} that have not been traversed. If such a node is found, the search process is restarted,

otherwise backtracking is performed by simply updating $t := t - 1$. Note that due to the special structure of Z, BitCLQ does not need to restore the set of compatible nodes during backtracking, in contrast to the update procedure (3.6) for the set C_S that is performed in FINDCLIQUE.

3.2.3 Example

As an illustration, consider the 3-partite graph that is shown along with its adjacency matrix M in Figure 3.2, where the partite 1 consists of nodes $\{1, 2, 3\}$, partite 2 contains nodes $\{4, 5, 6\}$, and partite 3 contains nodes $\{7, 8, 9\}$. BitCLQ is initialized by setting $S := \emptyset$, $\overline{B}_S := \{1, 2, 3\}$ and $Z_1 := \{111|111|111\}$. Since all the partites are of the same size, i.e. $|Z_{1,b}| = 3$ for all $b \in \overline{B}_S$, the one to branch on is chosen arbitrarily; assume that the first partite $Z_{1,1}$ is chosen for branching. The search process from this point restarts 3 times, each time adding one of the three nodes in $Z_{1,1}$. The first node to add to S is node 1, $Z_{1,1,1}$ is then set to 0, and Z_2 is subsequently created by performing logical AND operation with Z_1 and the corresponding row of the adjacency matrix M as operands:

$t := 1$,

$S := \{1\}$,

$Z_2 := Z_1 \wedge M(1) = \{011|111|111\} \wedge \{000|111|011\} = \{000|111|011\}$,

$\overline{B}_S := \{2, 3\}$.

As a result, the set Z_2 of nodes compatible with the partial solution $S = \{1\}$ contains nodes $\{4, 5, 6, 8, 9\}$. Since none of the partites in $\overline{B}_S$ is empty, the partial solution S is

Algorithm 3.1 BitCLQ(t)

1: $b_t \in \arg\min_b \{|Z_{t,b}| \mid b \in \overline{B}_S\}$

2: $i :=$ the first significant bit in Z_{t,b_t}

3: **repeat**

4: $\quad n_t :=$ the i-th node in b_t

5: $\quad Z_{t,b,i} := 0$

6: $\quad S := S \cup \{n_t\}$

7: $\quad$ **if** $|S| = k$ **then**

8: $\quad\quad Q := Q \cup S$

9: $\quad\quad S := S \setminus \{n_t\}$

10: $\quad$ **else**

11: $\quad\quad Z_{t+1,b} := Z_{t,b} \wedge M(n_t)$ for all $b \in \overline{B}_S$

12: $\quad\quad B_S := B_S \cup \{b_t\};\ \overline{B}_S := \overline{B}_S \setminus \{b_t\}$

13: $\quad\quad P :=$ number of partitions $Z_{t,b}$ with $|Z_{t,b}| > 0$, $b \in \overline{B}_S$

14: $\quad\quad$ **if** $|S| + P = k$ **then**

15: $\quad\quad\quad$ BitCLQ($t+1$)

16: $\quad\quad\quad S := S \setminus \{n_t\}$

17: $\quad\quad\quad B_S := B_S \setminus \{b_t\};\ \overline{B}_S := \overline{B}_S \cup \{b_t\}$

18: $\quad\quad$ **else**

19: $\quad\quad\quad S := S \setminus \{n_t\}$

20: $\quad\quad\quad B_S := B_S \setminus \{b_t\};\ \overline{B}_S := \overline{B}_S \cup \{b_t\}$

21: $\quad\quad$ **end if**

22: $\quad$ **end if**

23: $\quad i :=$ the first significant bit in Z_{t,b_t}

24: **until** $i \leq |V_{b'}|$

Figure 3.1: Pseudo-code for BitCLQ

promising and a new subproblem is created. The objective in the new subproblem is to find a $|\overline{B}_S|$-clique in Z_2. A node from $Z_{2,3}$ will be added to S (since $|Z_{2,3}| < |Z_{2,2}|$). The first node in $Z_{2,3}$ to add to the partial solution is node 8. The bit corresponding to node 8 is set $Z_{2,3,2} := 0$, and we have

$t := 2$,

$S := \{1, 8\}$,

$Z_3 := Z_2 \wedge M(8) = \{000|111|001\} \wedge \{111|001|000\} = \{000|001|000\}$,

$\overline{B}_S := \{2\}$.

Again, the partites in $\overline{B}_S$ contain at least 1 node (node 6) in Z_3. So the partial solution is promising, and a new subproblem is created. In the next step, node 5 is added to S:

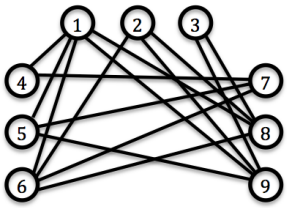
$t := 3$,

$S := \{1, 8, 6\}$.

At this point, since $|S| = k = 3$, i.e., a k-clique is found. To continue the search for other k-cliques, the last node in S is removed. BitCLQ searches $Z_{3,2}$ for another node that can be added to S. Since such a node does not exist, the algorithm backtracks: $t := 2$, node 8 is removed from S, and BitCLQ restarts with $S = \{1, 9\}$ as the partial solution.

3.3 Numerical Results

In order to illustrate the performance of the proposed method, the k-clique enumeration problem for k-partite graphs has been solved by BitCLQ and FINDCLIQUE



$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 3.2: A 3-partite graph and its adjacency matrix.

for randomly generated graph instances of several types. Both algorithms were implemented in C++ and ran on a 64-bit Windows machine with 3GHz dual-core processor and 4GB of RAM. It is worth noting that the original implementation of FINDCLIQUE algorithm by Grunert et al [30] relies on the use of `vectors` and `links` data types from the C++ standard template library (STL). In our experiments, we observed that by replacing the original data structure of vectors of lists with arrays, up to 300% improvement in FINDCLIQUE running time is achieved on the data sets used in our case study. The numerical results reported for the FINDCLIQUE algorithm are obtained using this "improved" implementation.

Our numerical experiments involve randomly generated instances of k-partite graphs of two types. The first set of instances consists of two groups: small-size instances and large-size instances. In the small-size instances, k-partite graphs are randomly generated with the number of partites in the range $k \in [3, 10]$. For each

Table 3.1: Average computational time (in seconds) to find all the k-cliques (#CLQ) contained in randomly generated k-partite graphs.

k	m	$\|V\|$	p	#CLQ	FINDCLIQUE	BitCLQ
3	100	300	0.1	1004	0.005	0.002
4	100	400	0.15	1124	0.008	0.002
5	100	500	0.2	1047	0.015	0.003
6	100	600	0.25	939	0.031	0.006
7	50	350	0.35	192	0.009	0.004
8	50	400	0.4	299	0.021	0.007
9	50	450	0.45	683	0.055	0.021
10	50	500	0.5	2672	0.176	0.071

value of k, the reported running times and the number of k-cliques in the graph are averaged over 10 instances. Table 3.1 shows the summary of the experimental results for this first group. The columns of the table show the number k of partites in the k-partite graph, the number m of nodes in each partite of the graph, the total number $|V|$ of nodes in the graph, the graph's density p, and the total number of k-cliques in the graph (#CLQ). The density parameter p is used for generation of the graphs, and is equal to the probability of an edge connecting two nodes from different partites: $\Pr\{(v_i, v_j) \in E\} = p$.

The second group include instances of larger size with the values of $k \in \{25, 50, 75, 100\}$. For each value of k in this group, 10 random instances of the k-partite graph have been generated and solved by FINDCLIQUE and BitCLQ. Table 3.2 summarizes the results of the experiments for this group. Since the graphs used in this set of experiments are rather large and the list of all k-cliques contained in them may not be found in a reasonable time, the solution process has been terminated after 200 seconds and

Table 3.2: Average number of k-cliques found in randomly generated instances of k-partite graphs after 200 seconds.

k	m	$\|V\|$	p	time	FINDCLIQUE	BitCLQ
25	40	1000	0.8	200	13,556,733	23,516,581
50	30	1500	0.9	200	800,369	1,032,111
75	30	2250	0.95	200	557,042,389	735,722,241
100	30	3000	0.95	200	348,416	365,799

the number of k-cliques found by each method was recorded. BitCLQ outperformed FINDCLIQUE in all cases.

The second set of experiments was conducted to compare the performance of BitCLQ with FINDCLIQUE on randomly generated instances of Multidimensional Assignment Problem (MAP). As shown in [38, 40, 49], high-quality solutions for randomized MAPs can be obtained as n-cliques in n-partite graphs that are constructed in a special way from the problem's data (in this case, n denotes the number of elements per dimension in a d-dimensional MAP). For MAPs with random iid costs, the resulting n-partite graph can be viewed as randomly generated with a certain density. The corresponding results are reported in Table 3.3, where n denotes the number of partitions in the graphs, and d is the number of dimensions d in the MAP.

Table 3.3: Average computational time (in seconds) needed to find the first n-clique in an n-partite graph corresponding to a randomized instance of the Multidimensional Assignment Problem with d dimensions and n elements per dimension.

n	d	m	$\|V\|$	p	BitCLQ	FINDCLIQUE
10	3	10	100	0.74	0.00	0.00
20	3	12	240	0.86	0.00	0.00
30	3	13	390	0.91	0.02	0.00
40	3	13	520	0.93	0.76	1.38
50	3	14	700	0.94	0.42	0.42
60	3	14	840	0.95	55.28	86.87
70	3	14	980	0.96	251.78	395.34
10	4	22	220	0.65	0.00	0.00
20	4	28	480	0.82	0.08	0.20
30	4	31	930	0.87	8.18	22.41
10	5	48	480	0.59	0.00	0.01
20	5	68	1360	0.77	13.29	28.23

CHAPTER 4
GRAPH PARTITIONING FOR THE DECOMPOSABLE COST MULTIDIMENSIONAL ASSIGNMENT PROBLEM

4.1 introduction

As was explained in chapter 2, given two sets V and W of cardinality n, the linear assignment problem seeks to find the least cost assignment of elements in set V to elements of set W (eq. (2.1)). It was also explained that when the number of sets, or equivalently *dimensions*, in an assignment problem is greater than two, the resulting problem is referred to as the Multidimensional Assignment Problem (MAP). As an example for an MAP consider the problem of assigning n jobs to n workers to n machines. This will be an instance of a 3-dimensional assignment problem (sec 2.1).

In this chapter, we are discussing heuristic methods for the decomposable cost MAP's. Decomposable cost MAP is a special case of MAP described in chapter 2 where in equation (2.7) the objective function will be replaced by:

$$Z^*_{d,n} = \min_{x\in\{0,1\}^{n^d}} \sum_{i_1=1}^{n} \cdots \sum_{i_d=1}^{n} c_{i_1\cdots i_d} x_{i_1\cdots i_d} \tag{4.1}$$

where the cost of each hypergraph $(i_1 \cdots i_d)$, denoted by $c_{i_1\cdots i_d}$, is obtained from:

$$c_{i_1\cdots i_d} = \sum_{e\in(i_1\cdots i_d)} w(e) \tag{4.2}$$

where e is an edge contained in the hyperedge and $w(e)$ is the cost (weight) of the edge e in the complete d-partite weighted graph representing the MAP. In other

words, the cost of each hyperedge in a decomposable cost MAP is the summation of the costs (weights) of the edges contained in the hyperedge.

Since a decomposable cost MAP can be converted to a regular MAP, some of approaches explained in this chapter will also apply to the regular MAP.

In the following sections, we will explain several methodologies for partitioning the graph representing the decomposable cost MAP, into several smaller disjoint subgraphs and show that the solutions from each subproblem (subgraph) can be recombined to provide upper and lower bounds to the original problem. The provided upper bound can be used in intelligent enumeration algorithms, such as branch and bound methods to prune the non-promising nodes in the branch and bound tree.

The idea that is used in the following section is that given an instance of a multidimensional assignment problem, any solution to the MAP with d dimensions and n elements in each dimension can be represented by a set of n disjoint d-cliques in the respective graph representation of the MAP. It is well-known that any instance of MAP with $m \geq 3$ is NP-complete. The number of solutions for such MAP grows exponentially with the number of dimensions d and factorially with the size of the dimension n[1]. In the reminder of the chapter, MAP will refer to decomposable cost multidimensional assignment problem unless otherwise stated.

[1]An MAP with number of dimensions d and size of each dimension n has $(n!)^d$ feasible solutions.

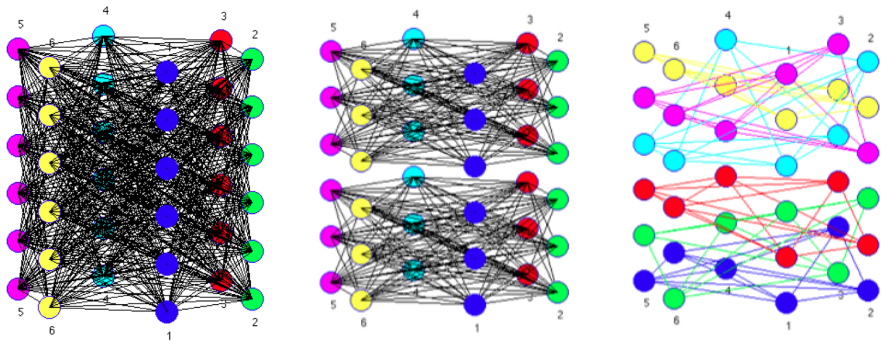
4.2 Element Partition

In this section, we will explain how an MAP instance can be partitioned along the elements in each dimension. We start by explaining simpler cases of dividing the elements of each dimension into two groups, and extend it to a case where only a single partition containing two elements from each dimension is considered. The remaining elements of each dimension are added to the said partition iteratively until it contains all the elements from all the dimensions.

4.2.1 Two disjoint element partition

Denote by MAP(d,n) an instance of an MAP with dimensionality (number of dimensions) d and cardinality (size of each dimension) n. The simplest way to partition the elements is to divide the elements in each dimension into two equally-sized groups. Each of the resulting problems will be an instance of $MAP(d, \frac{n}{2})$. In other words, the smaller problems will have the same number of dimensions but only half of the elements of the original problem in each dimension. Figure 4.1 shows the complete graph of an MAP(6,6) and how it can be partitioned into two subproblems. Figure 4.1(c) shows a feasible solution for each of the partitions. Subproblems can be solved by any available solver for MAP's. The optimal solution for each subproblem is a set of $\frac{n}{2}$ disjoint d-cliques. The final output of the two-disjoint element partitioning heuristic is the combination of the disjoint cliques in each subproblem. The cost of the solution obtained after the combination step is an upper bound on the cost of the optimal solution of the original problem as by partitioning we are disregarding a

portion of the feasible solutions of the original problem. Note that there are $[\binom{n}{r}]^d$ ways to partition nodes of each dimension into two disjoint partitions.



(a) (b) (c)

Figure 4.1: Partitioning of an MAP with $d = 6$ and $n = 6$: (a) A complete 6-partite graph MAP(6,6), (b) partitioning of the graph in two MAP(6,3) instances (c) 6 distinct cliques shown in each partitioning

The next theorem follows immediately:

Theorem 4.2.1. *For any two element partitions N_1 and N_2 such that $N_1 \cap N_2 = \emptyset$:*

$$Z^*[MAP(d, N_1)] + Z^*[MAP(d, N_2)] \geq Z^*[MAP(d, N_1 \cap N_2)]$$

where d is the set of dimensions and N is the set of elements, and Z^* the optimal value of a given MAP.

A full d-dimensional assignment problem has a complexity of $(n!)^{d-1}$, whereas each of the partitions have the complexity of $(\frac{n}{2})!^{d-1}$. Thus the complexity of solving

the partitioned problems is $2 \times (\frac{n}{2})!^{d-1}$, which is smaller than the complexity of the original problem. The two-disjoint element partitioning heuristic sacrifices the quality of the output to decrease the complexity of the problem. However we add another dimension of complexity to our problem due to the multiple ways of partitioning a single graph into two disjoint subgraphs.

4.2.2 Element augmentation

For an $MAP(d, n)$, the element augmentation method starts with a single partition containing all the dimensions but with only 2 elements from each dimension. The resulting subproblem is an instance of MAP(d,2). After solving the partial problem, another element from each dimension of the original problem is added to each of the dimensions in the subproblem and the new subproblem will be solved. This process repeats and new elements are added to the single partition until all elements from the original problem are considered in the partition, and an exact solution for the original problem is obtained . In each step, the solution obtained from the previous step can be used to obtain an upper bound for the new problem. The augmentation method is described in fig. 4.2.

An upper bound for the MAP(d,i) instance (line 5) is obtained by extending the $i-1$ disjoint cliques (the exact solution for the previous level) by adding a new disjoint clique containing the newly added elements.

The worst-case complexity of element augmentation method will be equal to:

Input: An instance of MAP(d,n)

1. Create a subproblem with 2 elements from each dimension.
2. Solve the resulting $MAP(d, 2)$ to optimality and obtain two d-cliques
3. **For** i = 3 **to** n
4. For each dimension select a new element and create an instance of MAP(d,n)
5. Obtain an upper bound for the MAP(d,i) instance by adding the d-clique formed by the new elements to the solution from the previous step.
6. solve the MAP(d,i) instance to optimality using the obtained upper bound from step 5.
7. **Next**

Figure 4.2: The pseudo-code for the element augmentation heuristic

$$T(d, n) = \sum_{i=2}^{n} (i!)^{d-1} \tag{4.3}$$

which is larger than the worst-case complexity of the original problem $(n!)^{d-1}$. This is due to the fact that to solve the MAP(d,n) using element augmentation method, intermediate subproblems MAP(d,2), ..., MAP(d,n) are solved separately. However, it is expected that the average complexity is reduced by the tighter upper bounds obtained from the intermediate subproblems.

4.3 Dimension Partition

An alternative method for element partitioning to solve MAP(d,n) is to partition the MAP along the dimension set. One approach in this category is *Two Disjoint Subgraphs*, the partitioning of the set of dimensions into two (usually) equally-sized sets to create two instances of $MAP(\frac{d}{2}, n)$. The other approach in this category is

the *Dimension Augmentation*; to start from two given dimensions and iteratively augmenting the subproblem with additional dimensions.

4.3.1 Two Disjoint dimension partition

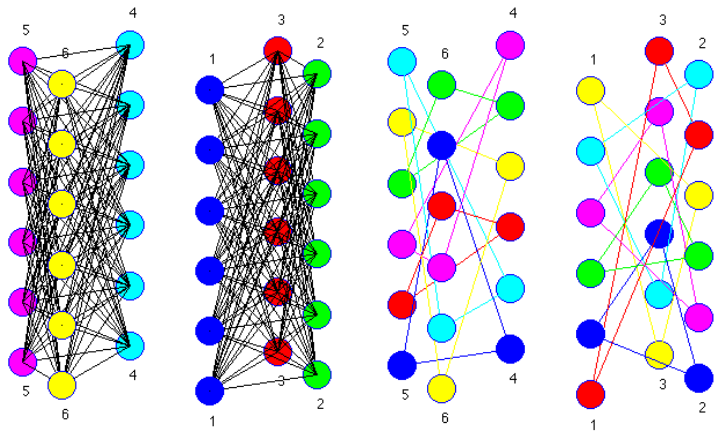
The idea in this approach is to convert a large d-dimensional assignment problem to two smaller $\frac{d}{2}$-dimensional assignment problems. Solving the smaller subproblems have the complexity of $2 \times (n!)^{\frac{d}{2}-1}$. However, in order to obtain a feasible solution for the original problem from the solution of each subproblem, a reconciliation step is necessary. As an example consider the two partitions for an MAP(6,6) displayed in fig. 4.3. This figure represents how the graph in 4.1(a) can be partitioned along its dimensions. The final solution to each subproblem contains 6 disjoint cliques (which is equal to the number of elements in each dimension of the original MAP) each of the size 3. A solution to the original problem should contain 6 disjoint cliques each of the size 6. For the reconciliation step, an LAP can be set up to determine the best assignment of the 6 disjoint cliques in the first set to the 6 disjoint cliques in the second set.

The next theorem follows immediately:

Theorem 4.3.1. *For any two nonempty dimension partitions D_1 and D_2, such that $D_1 \cap D_2 = \emptyset$,*

$$Z^*[MAP(M_1, N)] + Z^*[MAP(M_2, N)] < Z^*[MAP(M_1 \cup M_2, N)]$$

where M is the set of dimensions, N is the set of elements, and z^* is the optimal value of a given assignment problem.



(a) (b)

Figure 4.3: Dimension Partitioning of an MAP with $d = 6$ and $n = 6$: (a) The dimension partitioning of an MAP(6,6) (b) The disjoint cliques in each of the partitions.

The reason is that in the solutions obtained from the partitions, the edges necessary to convert the two groups of disjoint cliques into a feasible solution for the original problem are not considered. Consequently, any solution obtained before the conciliation step serves as a lower bound for the original problem.

4.3.2 Dimension Augmentation

The dimension augmentation method, which works in a similar fashion as the element augmentation method, starts with solving an assignment problem with only two dimensions considered. The remaining dimensions are iteratively added to the subproblem, based on a predetermined sequence. The initial subproblem with only 2 dimensions is an instance of a linear assignment problem that can be solved in

Procedure Dimension Augmentation Method

Input: an MAP(d,n), a sequence of dimensions $\sigma = \{d_1, d_2, \ldots, d_n\}$

1. Solve an LAP(2,n) on the first two dimensions d_1 and d_2
2. $\sigma := \sigma \setminus \{d_1, d_2\}$
3. **for** $i = 3$ **to** $|\sigma|$
4. Add d_i to the subproblem.
5. $\sigma := \sigma \setminus \{d_i\}$
6. Solve an LAP to determine an assignment of elements in d_i to the edges in the solution from the previous step
7. Solve the new subproblem with the solution obtained from line 5 as an upper bound.
8. **next** i

Figure 4.4: The pseudo-code for the dimension augmentation method

polynomial time. When the optimal solution of the initial subproblem is obtained, the third dimension from the predetermined sequence is added to the subproblem, to create an instance of MAP(3,n), and a feasible solution for the subproblem, which is suboptimal, is created by solving a linear assignment problem that determines the best assignment between the edges from the solution of the previous step and the nodes in the newly added dimension. The pseudo-code of the dimension augmentation method is shown in fig. 4.4. These steps are repeated, and new dimensions are added to the previous subproblem until all the dimensions in the original problem are included in the subproblem. At that point an exact solution for the original problem can be obtained.

4.4 Numerical results

In order to examine the performance of the heuristic and exact methods described in the current chapter, random instance of decomposable MAP were generated and solved by each method. The methods were implemented in C++ and run on a Windows machine with a 2.2 GHz CPU and 4GBs of RAM.

Instances of decomposable MAP with different cardinality (n) and dimensionality (d) were randomly generated. The cost of the edges for graphs were iid uniform in the range $[0, 1]$. Table 4.1 shows the result of the numerical experiments. The *ex* index is used for the exact method based on the existing method used in chapter 2, *ea* for the exact method based on the element augmentation, and *da* for the exact method based on the element augmentation. As can be observed, the exact methods based on augmentation methods are not very competitive with the existing methods. This can be due to the fact that the heuristics add another complexity to the process of decision making by introducing the orders by which elements or dimensions are added (augmented) to the base problem.

Table 4.1: Computational result and obtained cost for the exact and heuristic methods for MAP

n	d	$\|V\|$	$\|E\|$	$\|\mathcal{S}\|$	time_{ex}	cost_{ex}	time_{ea}	time_{da}
3	3	9	27	36	0.002	1.928	0.018	0.035
4	3	12	64	576	0.008	3.468	0.041	0.143
5	3	15	125	14400	0.015	3.322	0.112	0.22
6	3	18	216	518000	0.056	3.42	0.256	0.531
7	3	21	343	2.5e7	0.273	3.811	0.469	0.959
8	3	24	512	1.6e9	0.935	5.524	6.215	2.74
9	3	27	729	1.3e11	6.368	5.549	64.022	39.241
10	3	30	1000	1.3e13	9.251	5.502	16.012	147.304
11	3	33	1331	1.6e15	32.942	5.569	104.255	472.989
12	3	36	1728	2.3e17	122.465	6.095	1287.645	746.03
3	4	12	81	216	0.009	3.783	0.094	0.176
4	4	16	256	13824	0.351	4.707	2.339	3.363
5	4	20	625	1.7e5	0.873	5.014	6.999	15.882
6	4	24	1296	3.7e8	3.988	5.363	13.739	11.943
7	4	28	2401	1.3e11	67.049	6.711	318.358	1406.685
3	5	15	243	1296	0.046	4.44	0.137	0.655
4	5	20	1024	331776	4.599	1.423	14.807	57.556

CHAPTER 5
A BIT PARALLEL MAXIMUM CLIQUE ALGORITHM BASED ON MAXSAT

5.1 introduction

In this chapter we will review some of the methods available to solve the maximum clique problem (MCP) and finally offer a new method based on the algorithm proposed in Li & Quan [46]. The chapter concludes by presenting the result of computational experiments.

Consider an undirected graph $G = (V, E)$ where V is an ordered set of n vertices denoted by $\{v_1, \ldots, v_n\}$, and E is a set of m edges . A clique C in G is defined as a complete subgraph in G. The maximum clique problem (MCP) consists of finding the clique in G with the largest cardinality. The size of the largest clique in G is denoted by $\omega(G)$.

The methods to solve the maximum clique problem are divided into two groups: The heuristic methods and the exact methods. Among the heuristic methods, we have the *sequential greedy heuristics* that generate a maximal clique by iteratively adding or removing a vertex from a set that is not a clique, based on certain indicators associated with candidate vertices [33]. Sequential greedy heuristics only find a single maximal clique, and once one is found the algorithm stops. Clearly, the obtained solution is not guaranteed to be optimal. *Local search* heuristics expand this idea by searching the neighborhood of a maximal clique to possibly improve it. Depending on the type of neighborhood, different local search algorithms can be invented. k-interchange

heuristics are a well-known class of local search heuristics. The complexity of a k-neighborhood local search is $\mathcal{O}(n^k)$ where n is the size of the solution upon which the local search is being performed and k is the size of neighborhood [35]. Another type of heuristic methods for the maximum clique problem is the *stochastic methods*, usually inspired by a natural phenomenon, that try to avoid the local minima. Example of such methods are the simulated annealing method [11], genetic algorithm [59], scatter search [19], and ant colony optimization [24].

Branch and bound methods are one of the successful exact methods used to solve the maximum clique problem [69, 70, 65].

Let's use the following notations that are based on the definitions in chapter **??** to describe branch and bound methods for MCP: Q is the currently growing clique, Q_{max} is the largest maximal clique found so far (when the algorithm terminates, Q_{max} will contain a maximum clique of the graph), R is the candidate set, the list of nodes that are adjacent to all the nodes in Q and can be used to extend it.

A very basic maximum clique algorithm tries to list all maximal cliques in the graph by adding or removing nodes to/from Q, and repeating this loop until it finds a maximal clique that can be proved to be the maximum clique. Figure 5.1 shows *MaxClique*, a very basic maximum clique algorithm. In *MaxClique* search is done in the graph space. Sets Q and $Q_{\max}$ are globally maintained. *MaxClique* starts with $Q = \emptyset$ and $R = V$. At each iteration, a node is added to or removed from Q until it is verified that the largest maximal clique (the maximum clique) is found. The set of candidate vertices R at each iteration, comprises all the nodes that are adjacent to

all the nodes in Q:

$$R = \{v_i | (v_i, v_j) \in E, \forall v_j \in Q\}$$

In each iteration, *MaxClique* selects a node $v \in R$, and adds it to Q. *MaxClique* then computes the updated candidate set R' by removing nodes from R that are not adjacent to v. It then checks the pruning condition (line 5) by checking whether the current growing clique Q is probable to unset the best maximal clique so far Q_{max}. The pruning condition simply checks to see if the updated candidate set has enough number of nodes in it to possibly grow larger than $Q_{\max}$:

$$UP = |Q| + |R|$$

Later on, it will be shown how UP can be tightened to improve the quality of *MaxClique*.

As an example, consider the graph shown in figure 5.3. *MaxClique*, starts by ordering the nodes based on their degree in an ascending order. The result will be the sequence: $\{v_3, v_1, v_2, v_4, v_5, v_6\}$, which will be copied to set R. Node v_3 as the node with the highest degree is then added to the partial solution: $Q = \{v_3\}$. The nodes that are not adjacent to v_3 are removed from R: $R = \{v_1, v_5, v_6\}$. The upper bound for the current subproblem is set to be $UP = |Q| + |R| = 4$. For an intermediate step, if the calculated upper bound is smaller than the largest maximum clique found up to that point, the subproblem will be fathomed. Otherwise, the algorithm continues by branching deeper in the branch and bound tree and adding the node with the largest

Procedure *MaxClique* (R)
1. **while** $|R| > 0$
2. select node $v \in U$
3. $R := R \setminus \{v\}$
4. $R' := R \setminus C(v]$
5. **if** $|Q| + |R'| > |Q_{\max}|$ **then**
6. $Q := Q \cup \{v\}$
7. MaxClique (R')
8. $Q := Q \setminus \{v\}$
9. **else if** $|Q| > |Q_{\max}|$ **then**
10. $Q_{\max} := Q$
11. **wnd If**
12. **end while**

Figure 5.1: A very basic maximum clique algorithm

(original) degree to Q. In our example, *MaxClique* proceeds with adding node v_1 to Q:

$$Q = \{v_3, v_1\}.$$

Consequently, the nodes in R that are not adjacent to v_1 are removed and we will have:

$$R = \emptyset,$$

which implies that a maximal clique is found. At this point Q_{max} will be updated by copying the nodes in Q into it:

$$Q_{max} = \{v_3, v_1\}.$$

To important decisions in any maximum clique branch and bound are:

1. how to select the node to branch on
2. how to calculate the upper bound

Procedure MCQ(R,C)

```
1.    While R ≠ ∅
2.        select node v with maximum color C(v) from R ;
3.        R := R \ {v};
4.        if |Q| + C(v) > |Q_max| then
5.            Q := Q ∪ {v}
6.            if R' = R ∩ N_U(v) ≠ ∅ then
7.                generate a vertex coloring C of G(R')
8.                MCQ(R', C)
9.                Q =: Q \ {v};
10.           elseif |Q| > |Q_max| then Q_max := Q
11.       endif
12.   end while
```

Figure 5.2: MCQ algorithm

Most of the improvements on the basic maximum clique algorithm are based on modifications regarding these two decisions.

Östergård [53] proposed an alternative method for selecting the nodes from the candidate set that was different that the one proposed in 5.1. Given a graph $G(V, E)$, where V is a set of ordered vertices, i.e. $V = \{v_1, v_2, \ldots, v_n\}$, the algorithm iteratively finds the maximum clique for subgraphs $V_n = \{v_n\}, V_{n-1} = \{v_{n-1}, v_n\}, V_{n-2} = \{v_{n-2}, v_{n-1}, v_n\}, \ldots, V_1 = \{V\}$, the last subgraph being the original graph. The idea is that by solving the smaller subgraphs, better bounds can be found for the larger subgraphs.

The coloring problem in graph theory asks to find the minimum number of colors required to color the nodes in a given graph G such that no two adjacent nodes are assigned the same color. The minimum number of colors required to color the nodes of a graph is denoted by $\chi(G)$. Basically, the graph coloring problem (GCP) partitions

the vertices of graph G into the minimum number of independent sets.

One should note that not all the nodes in the candidate set R can in fact be added to the current clique Q simultaneously, and that the upper bound $(|Q| + |R|)$ is a loose overestimation. This upper bound can be improved if we subject R to a graph coloring algorithm. The output of the graph coloring algorithm is the minimum number of independent sets P needed to cover nodes in the induced subgraph $G[R]$. This value is called the Chromatic number of a graph G, and is denoted by $\chi(G)$. Obviously, a clique cannot contain more than a node from each independent set from R. Thus, a tighter upper bound can be achieved by using $|Q| + |P|$, where $P \leq R$ is a coloring of R and $|P|$ is the minimum number of colors needed to cover the nodes in graph G. However, it is well-known that the graph coloring problem is NP-hard. Thus, to establish a trade-off between the complexity of solving the graph coloring problem and tighter upper bound, instead of exact graph coloring methods, most branch and bound algorithms use a graph coloring heuristic instead. This idea is used in [69].

The resulting algorithm is obtained by replacing the pruning condition (line 5) in 5.1 with:

$$|Q| + |P| > |Q_{\max}|,$$

.

The algorithm in [69], hereafter referred to as MCQ, uses an approximate coloring method to assign a positive value (color) $C(v)$ to each vertex in V with the following

properties:

- If $(v_i, v_j) \in V$ then $C(v_i) \neq C(v_j)$.
- $C(v) = 1$, or if $C(v) = k > 1$, then there exists $v_1 \in N(v), v_2 \in N(v), \ldots, v_{k-1} \in N(v)$ in R such that $C(v_1) = 1, C(v_2) = 2, \ldots, C(v_{k-1}) = k - 1$.

The value $C(v)$ represents the largest possible extension in the size of the current growing clique Q, if node v is added to it. This graph coloring heristic method is used in several other maximum clique branch and bound methods.

In the beginning of the algorithm, MCQ sorts the nodes based on their degree in an descending form and copies them into set R. The relative order of the nodes in subsequent iteration of the algorithm is modified to be based on their $C(v)$ value. The idea, inspired by the observations in [53], is that since:

$$\omega(U) \leq \max\{C(v)|v \in U\}, \tag{5.1}$$

if $|Q| + \max\{C(v)|v \in R\} \leq |Q_{\max}|$ holds for a subproblem, all the nodes in the subproblem can be disregarded, and the whole subproblem can be fathomed (as opposed to a single node). The order of nodes in each subproblem is determined from the colors obtained in the previous recursion. Consequently, in MCQ, the next node to be added to Q is node v_i where:

$$i = \arg\max\{C(v_i)|(v_i) \in R\}.$$

In other words, the next node to add to Q, has the largest color number in R. In their experimental results they showed that their method outperforms the method

in [53] in most of the DIMACS benchmark instances. Ideally, it is desired to sort nodes descendingly based on their degree and assign color number to them such that a high-degree node gets a smaller color number. MCQ deviates from this by sorting nodes based on their color number.

As an example of how the graph coloring method in [69] works, consider the graph in fig. 5.3. For illustration purposes, lets denote by C_i all the nodes that are assigned color i (color class i). If we sort the nodes based on their degrees (in the beginning of the algorithm), we will obtain the set $V = \{v_3^{[3]}, v_1^{[2]}, v_2^{[2]}, v_4^{[2]}, v_5^{[2]}, v_6^{[1]}\}$, with the numbers in brackets being the degree of the nodes. The graph coloring method starts with node v_3, assigns it the color class 1, checks node v_1, assigns it the color class 2 (since nodes v_1 and v_3 are adjacent), and so on. The final results for the color classes will be: $C_1 = \{v_3, v_2\}$, $C_2 = \{v_1, v_4, v_6\}$, $C_3 = \{v_5\}$. Nodes are copied into the candidate set R in the same order that they appear in the color classes: $R = \{v_3, v_2, v_1, v_4, v_6, v_5\}$. Notice that

$$C(v_i) = k \iff v_i \in C_k.$$

Also note that the order of nodes in R has changed compared to the initial ordering based on degree. This will have a cascading effect on the coloring method. As the algorithm proceeds, it will deflect more from the ideal of assigning lower colors to nodes with higher degrees.

Also, since $\omega(G) \leq \Delta + 1$, with Δ being the largest degree among the nodes in G:

$$\Delta = max\{deg(v_i)|v_i \in V\},$$

MCQ never assigns a color larger than $\Delta + 1$ to any node.

For the graph in our example, node v_5 will be considered for inclusion in Q. The largest extension to the size of Q by adding v_5 is equal to $C(v_5) = 3$ and an upper bound for the current subproblem can be computed as $UB = |Q| + C(v_5) = 0 + 3 = 3$. Again, for an intermediate subproblem, if this UB is less than the size of an already-found maximal clique, this subproblem can be fathomed. Otherwise, MCQ proceeds with updating R, assigning color number to the nodes in the updated R, and adding a new node to Q.

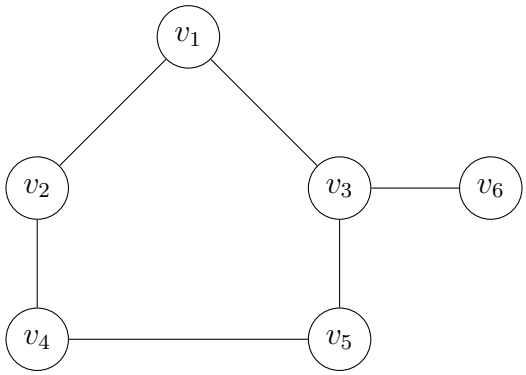


Figure 5.3: An imperfect graph with $\chi(G) = 3$ and $\omega(G) = 2$

In a new algorithm called MCS [70], Tomita et al. proposed two improvements over MCQ. The first improvement was the order by which nodes are fed into the approximate coloring method. As mentioned in the previous section, in MCQ, nodes

are ordered based on their colors in the previous iteration. In MCS, an approach is used to feed the vertices to the approximate coloring method based on their degree in the original graph.

They also proposed a recoloring method that works on top of the approximate coloring as follows: A new parameter called the *threshold color* $C_{th} := |Q_{max}| - |Q| + 1$ is defined. C_{th} is the minimum color for a node to be promising. If a node v is assigned a color higher than or equal to C_{th}, a recoloring procedure will be called to attempt to assign a lower color to this node and make it non-promising. Define C_i as the set of all nodes with color equal to i (color class i). The recoloring procedure on node v tries to find a vertex w in $N(v)$ such that $C(w) = k_1 \leq C_{th}$ with $|C_{k_1}| = 1$. If such w is found, then the procedure attempts to find k_2, where $k_1 < k_2 < C_{th}$, such that no vertex in $N(w)$ has color k_2. If such number k_2 is found, then the procedure changes the color of v and w such that $C(v) = k_2$ and $C(w) = k_1$. If any of this steps fails, $C(v)$ remains unchanged. But if all these steps are successfully performed, $C(v)$ will be changed to $k_1 \leq C_{th}$; thus it will be deemed non-promising and it is no longer necessary to search from v. MCS proved to be a very competitive method for the maximum clique problem.

In [36], in an algorithm referred to as MaxCliqueDyn, 2 modifications over [69] were proposed . The first modification was based on the observation that no node with a color number larger than C_{th} can save their original order in set R. This will make the order of nodes in R to be closer to their order obtained from sorting them based on their degree. The second modification was that they realized a tighter upper

bound can be obtained if the degree of nodes in subproblems in specific levels of the branch and bound tree were updated, based on the nodes existing in that particular subproblem, and fed into the graph coloring heuristic. Their computational result showed improvements over the MCQ method.

San Segundo etal. [65] proposed an exact bit-parallel method for the maximum clique problem. Their main contribution was to change the framework of the basic maximum clique branch and bound (Fig. **??**) by using bitsets as the data structure. Bitsets are explained in section 3.2.1. Using bitsets enables the process of updating the candidate set (line 6 in MCQ) to be performed more efficiently using the logical AND operator, which on average is done in one CPU instruction. The requirement for this is that the adjacency matrix of the graph as well as the candidate set R are stored in the memory of the computer using bitsets. They, later on, extended their algorithm in [65] to incorporate the recoloring method proposed in [70] in their bit-parallel framework. The resulting algorithm is one of the fastest maximum clique algorithms currently available.

Most branch-and-bound methods for the maximum clique problem use graph coloring heuristics to partition a graph into P independent sets to obtain an upper bound for a subproblem. For a perfect graph G, not only $\chi(G) = \omega(G)$, but also for any induced subgraph G' we will have $\chi(G') = \omega(G')$. So heuristic graph coloring methods seem suitable to obtain an upper bound for a subproblem in perfect graphs. However, in case of an imperfect graph, the upper bound obtained from the graph coloring method may not be very tight. Based on this concept, in [46], authors

propose a new encoding for the maximum clique problem into MaxSAT, and use the techniques used in MaxSAT solvers on top of graph coloring heuristics to improve the upper bound obtained. Their proposed algorithm proved to be one of the fastest maximum clique algorithms, able to solve several open DIMACS instances.

A boolean satisfiability problem (SAT) is the problem of determining whether the variables of a given boolean formula can be assigned in such a way as to make the formula evaluate to true. A SAT formula is usually represented in CNF[1], which is the representation of the formula as a conjunction (logical AND, denoted by $\wedge$) of clauses, where a clause is a finite disjuction (logical OR, denoted by $\vee$) of literals. A literal is variable that can be either true or false.

As an example for CNF consider the following:

- a
- $\bar{a}$
- $(a \wedge b)$
- $(a \wedge b) \vee (a \wedge \bar{c})$

A literal is satisfied if it is set to *true*, if it is in positive form, or *false*, if it is in negated form. A clause is satisfied if at least one if its literals is satisfied. A CNF is satisfied if all its clauses are satisfied simultaneously. If there is no assignment of variables to make the formula evaluate to true, the formula is *unsatisfiable*. As an example, consider $a \wedge b$ where $\wedge$ is the logical (=bitwise) AND operator. Boolean variables a and b are called *literals*. This formula is satisfiable, since one can assign

[1]Conjunction Normal Form

$a = true$ and $b = true$. However, $(a \wedge b) \wedge \overline{a}$ is not satisfiable, because there are no values for a and b that would simultaneously satisfy both $(a \wedge b)$ and $\overline{a}$.

Consider the following CNF formula:

$$(l_1 \vee l_2) \wedge (l_1 \vee l_3) \wedge (l_2 \vee \overline{l_4}) \tag{5.2}$$

First note that each clause is made of $\vee$ operators and the whole formula is a combination of conjunctions of clauses, which results in the formula being in CNF form. The CNF in (5.2) is satisfiable because we can assign $l_1 = true$, $l_2 = true$, $l_4 = false$. l_3 can take both true or false values because the second clause will be evaluated to true since $l_1 = true$.

A MaxSAT problem is a type of boolean satisfiability problem where the set of clauses are divided into *hard* and *soft* clauses. The goal in the MaxSAT problem is to find an assignment for the boolean variables that satisfy all the hard clauses while maximizing the number of satisfied soft clauses.

The traditional way to encode a Maximum Clique problem into a MaxSAT problem is to introduce a boolean variable x_i for each vertex $v_i \in V$. $x_i = 1$ i.f.f v is in the maximum clique. A hard clause $\overline{x_i} \vee \overline{x_j}$ is added for every pair of vertices v_i and v_j which are not connected ($(v_i, v_j) \notin E$). Every assignment satisfying hard clauses will be a clique[2]. The hard clauses prevent nodes that are not adjacent to be in the same clique. That is because satisfying a hard clause $\overline{x_i} \vee \overline{x_j}$ will require at least one of the variables x_i or x_j to be false, and thus they both cannot coexist in a clique. In

[2]Remember that a clique is a subset of nodes in V that are pairwise adjacent

order to find the largest clique contained in G a soft clause x_i is added to the set of clauses per each vertex in V. Maximizing the number of satisfied soft clauses while satisfying all the hard clauses results in a maximum clique.

As an example, consider the simple graph shown in figure 5.3. The Max-SAT based encoding for the maximum clique for this graph consists of 6 literals $x_1, x_2, \ldots, x_6$. The set of hard clauses are:

$$\{\overline{x_1} \vee \overline{x_4}, \overline{x_1} \vee \overline{x_5}, \overline{x_2} \vee \overline{x_3}, \overline{x_2} \vee \overline{x_5}, \overline{x_3} \vee \overline{x_4}, \overline{x_1} \vee \overline{x_6}, \overline{x_2} \vee \overline{x_6}, \overline{x_4} \vee \overline{x_6}, \overline{x_5} \vee \overline{x_6}\} \quad (5.3)$$

and the set of soft clauses are:

$$\{x_1, x_2, x_3, x_4, x_5, x_6\} \quad (5.4)$$

In [46], a new encoding for the maximum clique problem is proposed that is based on the result of the graph coloring heuristic.

Definition 5.1.1. *Let G be a graph partitioned into independent sets, the independent set based Max-SAT encoding of Max Clique is defined as follows:*

- *each vertex v_i in G is represented by a boolean variable x_i,*
- *a hard clause $\overline{x_i} \vee \overline{x_j}$ is added for each pair of non-adjacent vertices (v_i, v_j), and*
- *a soft clause is added for each independent set which is a logical or of the variables representing the vertices in the independent set (color class).*

The traditional way of encoding the maximum clique into MaxSAT is a particular case of Definition 5.1.1 where each independent set contains only one vertex. So if we partition[3] the graph in fig. 5.3 into 3 independent sets $\{v_1, v_4, v_6\}$, $\{v_2, v_3\}$, and $\{v_5\}$,

[3]Note that any graph coloring method can be used to do the partitioning.

an independent set based MaxSAT encoding for the maximum clique will have:

$$\{\overline{x_1} \vee \overline{x_4}, \overline{x_1} \vee \overline{x_5}, \overline{x_2} \vee \overline{x_3}, \overline{x_2} \vee \overline{x_5}, \overline{x_3} \vee \overline{x_4}, \overline{x_1} \vee \overline{x_6}, \overline{x_2} \vee \overline{x_6}, \overline{x_4} \vee \overline{x_6}, \overline{x_5} \vee \overline{x_6}\}$$

as the hard clauses and

$$\{\overline{x_1} \vee \overline{x_4} \vee \overline{x_6}\}, \{\overline{x_2} \vee \overline{x_3}\}, \{\overline{x_5}\}$$

as the soft clauses. Considering the hard clauses, any satisfied soft clause will not have more than one true literal, and the corresponding independent set for the soft clause will have only one vertex in the clique in any assignment satisfying the hard clauses. There are different graph coloring heuristics and each of them will possibly result in a different Max-SAT encoding of the maximum clique problem for a given (sub)graph. The authors conclude that the best method in terms of complexity and number of independent sets is the heuristic method explained in [69].

Unit Propagation (UP) is one of most commonly used techniques in Max-SAT solvers to detect disjoint inconsistent subset of soft clauses. A subset of soft clauses is called inconsistent if it results in a contradiction (empty clause) when considered with the set of hard clauses. A unit clause is a clause that contains a single literal. Given a CNF with unit clause l, the CNF will be satisfiable if and only if $l = true$. Considering this, unit propagation consists in the following two steps:

- all the clauses containing l will be removed, because they are automatically satisfied.
- $\bar{l}$ will be removed from any clauses containing it, because $\bar{l}$ cannot be used to satisfy the clause.

If after performing unit propagation, an empty clause is resulted, a contradiction has occurred, or equivalently, a *disjoint inconsistent subset* of soft clauses is detected. Also, if in the process, another unit clause is created, unit propagation can be performed with the newly found unit clause to find more contradictions. A literal l is called *failed* if when it is satisfied and $\bar{l}$ removed from other clauses, an empty clause is resulted. Every literal in soft clauses is checked to see if it is failed. Given a soft clause $c = l_1 \vee l_2 \vee \ldots \vee l_t$, if every l_i, $i = 1, 2, \ldots, t$ is a failed literal, the union of all the soft clauses used to produce an empty clause for every l_i, together with c, constitute an inconsistent subset.

They then state the following proposition:

Proposition 5.1.2. *Let $\omega(G)$ denote the cardinality of a maximum clique of a graph G. If G can be partitioned into k independent sets, and there are s disjoint inconsistent subsets of soft clauses in the independent set based MaxSAT encoding, then $\omega(G) \leq k - s$.*

Consider the independent set based Max-SAT encoding of the graph in the fig. 5.3. A traditional maximum clique solver would use $UB = 3$ as an upper bound on $\omega(G)$:

$$\omega(G) \leq 3$$

.

However, as can be shown, x_5 is a failed literal, and thus a tighter bound on $\omega(G)$ can be obtained:

If we set $x_5 = 1$ (a boolean variable set as 1 is interpreted as true), $\overline{x_5}$ will be removed from the hard clauses , resulting in three new unit clauses implying $x_1 = x_2 = x_6 = 0$. Then, x_1 and x_6 are removed from the first soft clause and x_2 from the second soft clause, making the second soft clause a unit clause. So x_3 and x_4 should be assigned 1 to satisfy their respective clause, making the hard clause $\overline{x_3} \vee \overline{x_4}$ empty. So x_5 is a failed literal and the three soft clauses used in the propagation to produce the empty clause constitute an *inconsistent subset*. This enables use to improve the upper bound from 3 to 2.

5.2 BitMSClique

In this section we will explain a new maximum clique, referred to as *BitMSClique* , which is based on the method in [46] and also uses the bitset structure similar to [65]. We will use the notation introduced in chapter 3.

The first modification in *BitMSClique* is to use the unit propagation exploited in [46] in a bit-wise framework proposed in [65]. Unit propagation will be used to assing a coloring number to individual nodes, rather than obtaining an upper bound on the size of $\omega(G)$. This enables us to consider only a part of graph G, when performing unit propagation, which is hoped to bring about improvement in the computational time. In [46], authors use a normal data structure for the maximum clique problem. The second modification in *BitMSClique* is the use of bitsets as data structures to update the candidate set in each iteration.

A color assigned to each vertex is nothing but the largest possible extension in the

size of the currently growing clique if nodes are to be added to the currently growing clique in a decreasing order of color numbers. In the new algorithm, an attempt will be done to use unit propagation simultaneously with the graph coloring heuristic, as opposed to after the graph coloring methods, to assign numbers to nodes in each subproblem. This may result in cases where the number assigned to the vertices are smaller than the index of the color class they belong to.

Specifically, in the new proposed algorithm, hereafter referred to as *BitMSClique* , the adjacency matrix of graph $G(V, E)$, $A(G)$, as well as the candidate set R are stored using a bitset data structure. Before the call to *BitMSClique* (R, C), All the bits in R are set to 1 and the numbers for the vertices are assigned using the method explained in [69].

Row i of $A(G)$, denoted by $A_i(G)$ is a bitset of size $|V| = n$. At each iteration, *BitMSClique* adds a vertex v from the candidate set R to the currently growing clique Q. Then, the new candidate set will be computed using bitwise operator AND:

$$R' = R \wedge A_v(G)$$

.

If the new candidate set $R' = \emptyset$ is empty (=contains no significant bits), a leaf node in the branch and bound tree is reached, or equivalently, a maximal clique in G is found. Q_{max} will be updated if the new maximal clique is larger. Otherwise, the algorithm backtracks. On the other hand, if $R' \neq \emptyset$, the *NewColoring* procedure is applied to assign colors (numbers) to each node in R'. After the numbers are assigned, *BitMSClique* will create a new subproblem and *NewMaxClique*(R', C') will

Procedure *BitMSClique* (R, C)

```
1.  while R ≠ ∅
2.      select a vertex v from R
3.      R := R \ v
4.      if |Q| + C(v) > |Q_max| then
5.          Q := Q ∪ {v}
6.          if R' = R ∩ A_v(G) ≠ ∅ then
7.              NewColoring(R', N')
8.              BitMSClique (R', N')
9.          elseif then
10.             Q_max := Q
11.         end if
12.         Q := Q \ {v}
13.     endif
14. end while
```

Procedure *NewColoring*(U, C)

```
2.  while R ≠ ∅
3.      select the first vertex from v ∈ U
4.      while (C_k ∩ C(v]) ≠ ∅
            k++
        end while
4.  C_k := C_k ∪ {v}
6.  C(v) = UP(v, C_1, ..., C_k) {perform unit propagation}
8.  end while
```

be called.

UP in the *NewColoring* procedure is where we perform the unit propagation to determine the final color number that should be assigned to a clique. The color classes $C_1, \ldots, C_k$ are used as the soft clauses in the process. The color assigned to a node after unit propagation is less than or equal to the color number assigned by the graph coloring heuristic.

Consider again the graph in fig. 5.3. *BitMSClique* starts by setting $Q = \emptyset$ and $R = 111111$. The sequence of nodes sorted based on their degree will be: $\{v_3, v_1, v_2, v_4, v_5, v_6\}$. Based on this sorted order, the first bit in R is corresponds to node v_6, the second bit corresponds to node v_5 and so on. The adjacency matrix can also be reconstructed (permuted in rows and columns) to reflect this new mapping.

BitMSClique then proceeds by performing the graph coloring heuristic to assign color numbers to nodes in R. The resulting colors will be:

$C(v_3) = 1$;

$C(v_1) = 2$; since $(v_1, v_3) \in E$

$C(v_2) = 1$;

$C(v_4) = 2$; since $(v_2, v_4) \in E$

$C(v_5) = 3$; since $(v_3, v_5) \in E$ and $(v_4, v_5) \in E$

$C(v_6) = 2$; $(v_3, v_6) \in E$

However, using the *NewColoring*, The numbers assigned to each node will be:

step 1) $C(v_3) = 1$;

Step 2) $C(v_1) = 2$;

Step 3) $C(v_2) = 1$;

Step 4) $C(v_4) = 2$;

Step 5) $C(v_5) = 2$;

Step 6) $C(v_6) = 2$;

Note that the *NewColoring* procedure assigns $C(5) = 2$, as opposed to $C(5) = 3$ for the traditional coloring. The reason behind it is that if we create the MaxSAT instance of the nodes that are numbered up to step 5 the soft clauses will be:

$$\{(v_3 \vee v_2), (v_1 \vee v_4), v_5\}$$

with hard clauses:

$$\{(\overline{v_1} \vee \overline{v_4}), (\overline{v_1} \vee \overline{v_5}), (\overline{v_2} \vee \overline{v_5}), (\overline{v_2} \vee \overline{v_3}), (\overline{v_3} \vee \overline{v_4})\}$$

After setting $v_5 = 1$, we should have $v_1 = v_2 = 0$, which converts the first soft clause into the unit clause (v_3, and the second soft clause into the unit clause (v_4). Repeating the unit propagation again with node v_4 and setting it to *true* forces us to set $v_3 = false$ which creates an empty clause. Thus, v_5 in this encoding is a failed literal and its assigned number will be 2, instead of 3.

BitMSClique then proceeds by adding node v_6 to Q, and updating the candidate set by performing a logical AND operation:

$$R' = 111111^0 01000 = 001000.$$

In the next iteration, node v_3 (the only significant bit in R') will be added to Q and a maximal clique is found v_6, v_3. At this point, *BitMSClique* can terminate as it is obvious from the numbers assigned to nodes that a maximal clique if size 2 is also maximum.

Table 5.1: Time spent to find the maximum clique in random graphs of different size

$\|V\|$	CLQ	BitMSCLQ	MaxCliqueDyn
20	5.35	**0.001**	0.013
50	7.4	**0.005**	0.017
100	9.15	0.076	**0.013**
150	10.2	0.472	**0.013**
200	11	1.985	**0.027**
250	11.3	7.918	**0.061**
300	12.05	21.527	**0.121**
350	12.35	83.532	**0.28**
400	12.9	180.016	**0.572**
450	13.05	361.821	**1.091**
500	13.1	669.32	**2.3**

5.3 Experimental results

BitMSCLQ was implemented in C++ and used to solve random graphs of different size. The code was run on a windows machine with a 2.2 GHz CPU and 4 GB of ram. For the sake of comparison, the same problems were solved by MaxCliqueDyn [36]. Table 5.1 shows the computational time to solve the graphs of each size. For each size reported in the table, the computational time reported is averaged over 20 instances.

BitMSCLQ outperforms MaxCliqueDyn in instances of size up to 50. But its performance degrades significantly as the size of the graph increases beyond 50. Problems in the SAT family have been the subject of extensive research and studies. Unfortunately, the implementation of algorithms to solve SAT problems are not usually publicly available. This was the case for the method in [46] also. The implementations usually contain considerable amount of optimization, without knowing which, replicating or contributing to the result of an algorithm will be difficult.

CHAPTER 6
RISK-AVERSE MAXIMUM CLIQUE PROBLEM

6.1 Introduction and motivation

In this chapter, we study the minimum-risk, maximum clique problem. An inherent part of most networks in real life is the uncertainties encountered in the network components. These uncertainties are observed in several shapes and can lead to infeasibility/suboptimality of the otherwise optimal network decisions, and incur unexpected costs or losses. Stochastic factors in network optimization problems have been the subject of several studies in different fields such as operations research, industrial engineering, computer science, and geographic information systems.

Although most traditional stochastic network models consider stochasticity associated to network arcs, in this work, we consider network problems with stochastic factors associated with the nodes in the network. We will present a mathematical formulation of the *risk-averse maximum clique* problems, and then show how in certain situations, the optimal solution of the problem corresponds to a maximal clique in the underlying graph representing the network. We will also provide a combinatorial optimization approach to solve the problem.

Give a graph $G = (V, E)$ with the node set V and the edge set E, where each node $i \in V$ has an associated random variable X_i representing cost or loss, with a known joint distribution of X_i's, with a given risk measure ρ, the problem of finding the minimum-risk subgraph of G with a prescribed property $\mathcal{Q}$ can be depicted by

the following mathematical program:

$$\begin{aligned} \min_{S \subseteq V(G),\, w} \quad & \rho\Big(\sum_{i \in S} w_i X_i\Big) \\ \text{s. t.} \quad & \sum_{i \in S} w_i = 1 \\ & w_i \geq 0, \quad i \in V \\ & S[G] \in \mathcal{Q}_G, \end{aligned} \tag{6.1}$$

where $S[G]$ is the subgraph induced [1] by a subset S of nodes $V(G)$, and $\mathcal{Q}_G$ is the set of all subgraphs of G with the desired property $\mathcal{Q}$. In this chapter, we are interested in *complete subgraphs*, or *cliques*, in G:

$$\mathcal{Q}_G = \{S \subseteq V(G) \mid \forall i, j \in S : (i, j) \in E(G)\}. \tag{6.2}$$

The objective function in 6.1 represents the risk associated to the subgraph. The variables w_i in (6.1) represent the weights based on which vertices of the minimum-risk induced subgraph of G are selected.

The presence of wieghts in 6.1 can be motivated in a network where each node contributes to the overal risk of the system. As an example, consider a network with nodes representing sensors that output information of uncertain quality. The edges connecting sensors (nodes) enable them to share information and potentially enhance the output of the system. In such a system, the weights can represent a source (e.g. money or energy) that is needed to keep a sensor alive, such that the risk of information loss in the system is minimized. This situation can be formulated as

[1] An induced subgraph of $G(V, E)$ has the same set of edges that appear in G over $V' \subseteq V$.

the aforementioned risk averse maximum clique problem.

The risk-averse (or minimum risk) maximum clique problem is introduced in [63]. The problem seeks to find the largest clique of certain property, contained in an undirected graph that represents a network with random variables, with a known joint distribution associated to each vertex. Some of the studies in the literature, that also consider stochasticity associated to nodes are discussed here briefly. The influence of long term demand uncertainties in a network system is confirmed in a study by Ukkusuri and Mathew [71]. They compared traffic network design problems (TNDP) with uncertainties with their deterministic counterpart, and proposed a method based on Genetic Algorithm to provide a robust solution. Atamturk and Zhang [5] developed a novel methodology to solve network problems entailing uncertain network demands, based on a two-stage stochastic optimization model. In their method, decisions were deferred until the demand was materialized. Glockner and Nemhauser [26] studied a dynamic network flow problem with random capacities represented on arcs. Their method is based on a multistage stochastic linear program. They propose other applications focusing on cases where flow through the network is affected by uncertainties attributed to arcs. Several studies examined the effects of stochastic arc failures on networks. Aneja et al. [2] analyzed flow patterns that maximize residual flow under probabilistic arc failure. Verweij et al. [73] used a sample average approximation method to solve several two-stage stochastic routing problems subject to arc failures and unexpected delays. Boginski et al. [9] and Sorokin et al. [67] proposed a mathematical programming approach minimizing flow losses through

a network by capturing the impact of probabilistic arc failures relative to conditional expectation of worst-case outcomes.

6.2 Risk measures in stochastic programming

Risk is a subjective assymetric phenomenon involving exposure and uncertainty. Subjective, in that, different methodologies define the same situation to have different risks. assymetric, because it is considered for losses or costs only. The first step to recognize risk is to quantify it. Given the probability space $(\Omega, \mathcal{F}, ¶)$, the risk measure associated with a random outcome X is denoted by $\rho(X)$ and is defined as a mapping $\rho : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is a space of bounded $\mathcal{F}-$measurable functions $X : \Omega \mapsto \mathbb{R}$. In what follows, it is assumed that X represents a cost or a loss, whereby its larger realizations are considered "riskier".

Risk measures are usually selected based on the context and the application. The context in which the risk measure is used in usually calls for the presence of certain properties of the risk measure. This however, may lead to inappropriate selection of risk measures. A notorious example is the VaR (Value-at-Risk), which is widely used in financial institutions. VaR with the confidence level $\alpha \in (0, 1)$is defined as:

$$\mathrm{VaR}_\alpha(X) = \inf\{\eta \mid ¶[X \leq \eta] \geq \alpha\}, \tag{6.3}$$

which is equal to the α-quantile of the loss distribution. VaR is an estimation of the loss with a confidence level for a certain time period. The most significant drawback of using VaR to measure risk is that VaR is a non-convex function and is

non-sub additive (explained later), thus, does not account for the risk reduction via diversification which is a fundamental principle of risk management.

Recent studies in risk theory, pioneered by Artzner et al. [4] identified four properties, or axioms, that a "good" risk measure should possess. These four properties are: *monotonicity*, *sub-additivity*, *positive homogeneity*, and *transitional invariance*. Any risk measure possessing these properties is identified as a *coherent risk measure*. Monotonity is expressed mathematically as:

(**A1**): $X \leq 0 \Rightarrow \rho(X) \leq 0$ for all $X \in \mathcal{X}$,

which ensures that based on the selected risk measure, lower losses have lower risks.

Sub-additivity, which is one of the fundamental pillars of risk management theory, expresses the idea of risk reduction through diversification. In mathematical terms, sub-additivity is formulated as:

(**A2**): $\rho(X+Y) \leq \rho(X) + \rho(Y)$ for all $X, Y \in \mathcal{X}$

Based on the positive homogeneity property, the risk for a loss, scales linearly for any positive weight:

(**A3**): $\rho(\lambda X) = \lambda \rho(X)$ for all $X \in \mathcal{X}$ and $\lambda > 0$

and finally, the transitional invariance property implies that constant changes in X affect the associated risk by the same amount:

(**A4**): $\rho(X+a) = \rho(X) + a$ for all $X \in \mathcal{X}$ and $a \in \mathbb{R}$

Note that as long as the positive homogeneity property holds, the sub-additivity property can be substituted by the *convexity* axiom:

(**A2′**) *convexity:* $\rho(\lambda X + (1-\lambda)Y) \leq \lambda\rho(X) + (1-\lambda)\rho(Y)$ for all $X, Y \in \mathcal{X}$, $\lambda \in [0,1]$

The convexity property, allows for the utilization of efficient mathematical methods to optimize the risk using a coherent measure.

It is important to notice that the above axiomatic definition for coherent risk measures does not provide a functional form of coherent risk measures. Krokhmal [41], proposed a representation of coherent measures of risk as a convolution of some function $\phi : \mathcal{X} \mapsto \mathbb{R}$ that satisfies (A1)-(A3), which is a lower semicontinuous function for which $\phi(\eta) > \eta$ for all real $\eta \neq 0$ holds. For such function, it is shown that the following stochastic programming problem, which is convex, has an optimal value, and such value is a proper coherent measure of risk:

$$\rho(X) = \min_{\eta \in \Re} \ \eta + \phi(X - \eta) \tag{6.4}$$

Any coherent risk measures which admit representation (6.4) can be included in mathematical programming models in the form of objective function or as a constraint [41]. An instance of such risk measures widely used in stochastic optimization models is the Conditional Value-at-Risk (CVaR) [61, 62]:

$$\mathrm{CVaR}_\alpha(X) = \min_{\eta \in \mathbb{R}} \ \eta + (1-\alpha)^{-1}\mathbb{E}(X-\eta)^+, \quad \alpha \in [0,1], \tag{6.5}$$

where $X^+ = \max\{0, X\}$. Conditional Value-at-Risk with confidence level α is denoted by $\mathrm{CVaR}_\alpha(X)$, and gives the average loss beyond the $\mathrm{VaR}_\alpha(X)$ level:

$$\mathrm{CVaR}_\alpha(X) = \mathbb{E}[X \mid X \geq \mathrm{VaR}_\alpha(X)]. \tag{6.6}$$

It is noteworthy that equation 6.6 only holds in certain special cases, such as when the loss function X is continuous. Interested readers are referred to [62] for more details on CVaR for loss functions with general distributions.

Of special interest to us in this chapter is the case when the loss function X is discrete and is defined by a scenario set $\mathcal{N}$, where the probability of each scenario occuring is denoted by p_s:

$$\P\{X = X_s\} = p_s.$$

With a discrete loss function, the optimization problem in 6.5 transforms into a stochastic programming problem of the form:

$$\begin{aligned} \min \quad & \eta + (1-\alpha)^{-1} \sum_{s \in \mathcal{N}} p_s t_s \\ \text{s. t.} \quad & t_s \geq X_s - \eta, \quad s \in \mathcal{N} \\ & t_s \geq 0, \quad s \in \mathcal{N}, \end{aligned} \tag{6.7}$$

where t_s is an auxiliary variable associated with scenario $s \in \mathcal{N}$ that guarantees the equivalence of two objective functions.

Through the rest of this chapter, and in the numerical experiments, CVaR is used as the risk measure denoted by ρ. The general approach introduced here is applicable to a broad class of coherent risk measures of the form (6.4).

6.3 Risk-averse maximum clique problems

In this section we first provide the descriptive definition of minimum-risk maximum clique problem (6.1), and then explain a special case of the general case, namely, the *isolated risk exposure* and the *neighbor-dependent risk exposure*, and ultimately present the mathematical formulation for the risk-averse maximum clique problems with isolated risk exposures.

Denote by $\mathcal{R}(S)$ the risk of selecting subgraph $G[S]$ of a given graph G:

$$\mathcal{R}(S) = \min \left\{ \rho\big(X_G(S;w)\big) \;\middle|\; \sum_{i\in S} w_i = 1; \;\; w_i \geq 0 \;\; \forall\, i \in S \right\}, \tag{6.8}$$

where $\rho(X)$ represents a (coherent) measure of risk, and $X = X_G(S;w)$ denotes the loss function defined over a subset S of nodes in G, which also is a function of the weights w of nodes in S. An explicit example for this is (6.1), with a loss function that has a weighted-sum form:

$$X_G(S;w) = \sum_{i\in S} w_i X_i.$$

A broad set of mathematical problems can be defined by focusing our attention to subgraphs in G with a prescribed property $\mathcal{Q}$, all of which can be represented in the form:

$$\min \big\{ \mathcal{R}(S) \;\big|\; S[G] \in \mathcal{Q}_G \big\}. \tag{6.9}$$

Examples for $\mathcal{Q}_G$ can be the set of all independent sets in G:

$$\mathcal{Q}_G = \{S \subseteq V(G) \mid \forall i, j \in S : (i, j) \notin E(G)\},$$

or the set of all the paths contained in graph G between nodes s and t:

$$\mathcal{Q}_G = \{S \subseteq V(G) \mid S = \{s \equiv i_0, i_1, \ldots, i_{n-1}, i_n \equiv t\};\ (i_{k-1}, i_k) \in E(G),\ 1 \leq k \leq n\}.$$

When expanded, (6.9) can be written as:

$$\begin{aligned} \min_{S \subseteq V(G),\, w} \quad & \rho\big(X_G(S; w)\big) \\ \text{s. t.} \quad & \sum_{i \in S} w_i = 1 \\ & w_i \geq 0, \quad i \in V \\ & S[G] \in \mathcal{Q}_G. \end{aligned} \tag{6.10}$$

In the current chapter, we are mainly concerned with the set of complete subgraphs, or cliques, in G, as defined in (6.2). Under such case, (6.10) will depict the general formulation of the risk-averse maximum clique problem.

Special cases of the loss function $X_G(S; w)$ in (6.10) include situations where the risk exposure of node i depends on both its own loss profile X_i as well as losses of its neighbor nodes. The overal risk of a subset S is then a function of the risk of individual nodes and its adjacent nodes. This special case can describe many real-life applications, such as the ones observed in financial contexts considering inter-bank loans, which heavily exposes counterparties. However, here we only consider the cases when the risk exposures of individual nodes are *isolated*, and unaffected by the risk

profile of their neighbors. In other words, stochastic factors of nodes do not impact the risk at their neighbors.

In the next section, we will proved the mixed integer programming formulation of the risk averse maximum clique problem with isolated stochastic effects. For the purpose of correctness, we will assume the risk measure ρ to be selected as the Conditional Value-at-Risk with the confidence level α, $\rho(X) = CVaR_{\alpha}(X)$. Further, we assume that X_i values, the losses associated with node $i \in V$, have a discrete joint distribution, depicting as a 2-dimensional scenario matrix $\mathcal{N}$, where X_{is} represents a stochastic factor X_i under scenario $s \in \mathcal{N}$.

6.3.1 Risk-averse maximum clique problem with isolated risk exposures

According to the discussion above, the loss function $X_G(S; w)$ corresponding to isolated risk exposures is defined simply as a weighted sum of losses X_i among selected nodes $i \in S$:

$$X_G(S; w) = \sum_{i \in S} w_i X_i. \tag{6.11}$$

By introducing binary decision variables x_i, $i \in V$, such that

$$x_i = \begin{cases} 1, & i \in S, \\ 0, & i \notin S, \end{cases}$$

where S is the desired subgraph, the *risk-averse maximum clique problem with isolated risk exposures* can be formulated as a mixed integer programming problem of form

$$\min \quad \rho\Big(\sum_{i\in V} w_i\, x_i\, X_i\Big) \tag{6.12a}$$

$$\text{s. t.} \quad \sum_{i\in V} w_i = 1 \tag{6.12b}$$

$$w_i \le x_i, \quad \forall i \in V \tag{6.12c}$$

$$x_i + x_j \le 1, \quad \forall (i,j) \in \overline{E} \tag{6.12d}$$

$$x_i \in \{0,1\}, \;\; w_i \ge 0, \quad \forall i \in V. \tag{6.12e}$$

Constraint (6.12c) ensures that weights w_i can be non-zero only for the vertices i that are included in the solution S, while constraint (6.12d) maintains that the set of selected nodes forms a complete subgraph, or a clique. Observe that due to the presence of constraint (6.12c) the nonlinearity in the objective function (6.12a) attributed to the products $w_i x_i$ can be eliminated by replacing $w_i x_i$ with just w_i, so that the objective of (6.12) takes the form

$$\rho\Big(\sum_{i\in V} w_i X_i\Big).$$

When the joint distribution of stochastic factors X_i, $i \in V$, is given by scenario set $\{X_{is}\}_{s\in\mathcal{N}}$, and risk measure ρ is chosen as CVaR_α, the risk-averse maximum clique problem (6.12) reduces to the following 0–1 mixed integer stochastic programming

problem

$$\min \quad \eta + \frac{1}{1-\alpha} \sum_{s \in \mathcal{N}} p_s t_s \tag{6.13a}$$

$$\text{s. t.} \quad \sum_{i \in V} w_i = 1 \tag{6.13b}$$

$$w_i \leq x_i, \quad \forall\, i \in V \tag{6.13c}$$

$$x_i + x_j \leq 1, \quad \forall\, (i,j) \in \overline{E} \tag{6.13d}$$

$$t_s \geq \sum_{i \in V} w_i X_{is} - \eta, \quad \forall\, s \in \mathcal{N} \tag{6.13e}$$

$$x_i \in \{0,1\}, \;\; w_i \geq 0, \;\; \forall\, i \in V; \;\; t_s \geq 0 \;\; \forall s \in \mathcal{N}, \tag{6.13f}$$

where p_s is the probability of scenario $s \in \mathcal{N}$, i.e.

$$\P\Big\{ \bigcap_{i \in V} X_i = X_{is} \Big\} = p_s, \quad s \in \mathcal{N},$$

and, naturally, one has $\sum_{s \in \mathcal{N}} p_s = 1$.

Finally, we show that the adopted definition (6.8) of risk $\mathcal{R}(S)$ for subgraph S and the chosen loss functions $X_G(S; w)$ of form (6.11) is consistent with the sub-additivity property of coherent risk measures. Namely, we demonstrate that the following is true.

Proposition 6.3.1. *Consider definition* (6.8) *of risk for subset S of vertices in graph $G = (V, E)$, where each vertex $i \in V$ is associated with a random element X_i. If risk measure ρ in* (6.8) *is coherent, and the loss function associated with selecting $S \subseteq V$ is given by* (6.11)*, then risk $\mathcal{R}$ satisfies*

$$\mathcal{R}(S') \leq \mathcal{R}(S) \quad \textit{for all} \quad S' \supseteq S. \tag{6.14}$$

Corollary 6.3.2. *Proposition* (6.3.1) *implies that an optimal solution of risk-averse maximum clique problem* (6.12) *represents a* maximal *clique of the underlying graph* G.

6.4 A combinatorial approach to solve the maximum clique problem with isolated risk exposures

As was noted in corollary (6.3.2), the optimal solution of the risk-averse maximum clique problem is a maximal clique in the underlying graph representing the network. In this section we propose and explain a combinatorial approach to solve the risk-averse MCP. For the sake of brevity, we refer the readers to section 6.3 for preliminary definitions.

The proposed method, hereafter referred to as RAMCQ, is based on general branch-and-bound method for solving the maximum clique problem. For a graph $G(V, E)$, RAMCQ tries to find all the maximal cliques contained in G, until one is found that is guaranteed to have the smallest risk. RAMCQ maintains 2 sets for each node of the branch and bound tree. The first set, denoted by Q, is a partial solution to the problem and contains the node that construct a clique. In other words, all the nodes in Q are pair-wise adjacent. The second set maintained by RAMCQ is the set of candidate node, denoted by R, which contains the nodes that are adjacent to all the nodes in Q, and can be used to extend Q:

$$R = \{v_i | (v_i, v_j) \in E, \forall v_j \in Q\}$$

The pseudo-code for RAMCQ is given in Fig. 6.1. The input to RAMCQ is the risk-averse maximum clique problem formulation (denoted in the pseudo-code by P), and the graph underlying it. RAMCQ is initialized with $Q = \emptyset$ and $R = V$ for the root node. Hereafter, to avoid confusion, the nodes in the sets Q and R are referred to as elements, to distinguish them from the nodes of the branch and bound tree. At each step of RAMCQ, it is determined, as explained later, whether the node is promising or not. If the node is deemed promising, an element $v \in R$ of the respective node in BnB tree is selected and branched on. The next subproblem will have $Q' = Q \cup \{v\}$ and $R' = R \cap \Gamma(v)$ as its partial solution and candidate set. If a node of the branch and bound tree is deemed non-promising, RAMCQ backtracks from the current node of the BnB tree to its parent node, and then tries to branch on another element in the candid set. Backtracking also occurs if all the elements in the candid set of a particular node of the BnB tree are branched on.

RAMCQ uses a relaxation of the risk-averse maximum clique problem to calculate a lower bound on the value of the optimal risk of a BnB tree node. Specifically, we relax constrains 6.13d and 6.13f, and the resulting mathematical formulation, as is shown at (6.15), is solved to obtain an overestimation of the minimum risk the current subproblem can lead to.

$$LB_{rel} = \min \quad \eta + \frac{1}{1-\alpha} \sum_{s \in \mathcal{N}} p_s t_s \tag{6.15a}$$

$$\text{s. t.} \quad \sum_{i \in V} w_i = 1 \tag{6.15b}$$

$$t_s \geq \sum_{i \in V} w_i X_{is} - \eta, \quad \forall\ s \in \mathcal{N} \tag{6.15c}$$

In the pseudo-code, the input to the RAMCQ are the risk-averse problem (P), and the graph representing it. The variables r^* and Q^* represent the value of the optimal risk, and the risk-averse maximum clique contained in G. Also, r^*, Q^* and P are globally defined. The expand subprocess is called after the initialization. This subprocess is where the BnB traversal occurs. Expand is a recursive function, and recalls itself (line 16) until a leaf node in the BnB tree (a maximal clique) is reached (line 17), at which point, the best solution of the problem will be updated in necessary. Expand returns to the previous call (backtracks), if all the elements of R in a particular node of the BnB are examined.

6.5 Numerical experiments

In order to evaluate the performance of RAMCQ, random instances of risk averse maximum clique problem with isolated risk exposure are generated and solved by RAMCQ and compared to CPLEX. It is a well-known fact that both the maximum clique problem and the maximal clique enumeration problem are NP-hard. Consequently, the risk-averse maximum clique problem is NP-hard as well. Generally, due to the presence of stochastic factors, the largest size of graphs that can be solved for

```
1. def RAMCQ(P, G(V, E)):
       // Output: risk-averse maximum clique in G, denoted by Q*
       Initialize: Q = ∅, R = V, r* = ∞, Q* = ∅
       call Expand(R, Q)
5.     return Q*

def Expand(R, Q):
       while |R| > 0:
           select v ∈ R
           R = R\v
10.        rel = relaxation(P, Q, v)
           LB = rel.solve()
           if (LB < r*):
               R' = R ∪ Γ(v)
               Q' = Q ∪ {v}
15.            if |R'| > 0:
                   call Expand(R', Q')
               else
                   Q* = Q
                   r* = LB
               fi
20.        fi
       wend
```

Figure 6.1: Pseudo-code for RAMCQ

the risk averse problem is smaller than graphs that can be solved for the maximum clique problem.

The graphs generated are Erdos-Renyi graphs [23] $G(V, p)$ where every edge is independently formed with a prescribed probability p. Random scenario data corresponding to each vertex $i \in V$ were generated according to a uniform distribution over an interval $[-0.5, 0.5]$.

RAMCQ was implemented in C++ and run on a windows machine with a 2.2 GHz CPU and 4 GBs of RAM. To solve the mixed integer program, CPLEX 12.2 with the C++ API was used and run on the same machine.

Table 6.1 summarizes the results obtained from solving graphs of different sizes and densities under specified number of scenarios. For each row of the table, 20 instances were generated and solved by both methods, and the values reported are averaged over all instances. A time limit of 1000 seconds was considered for both methods, and an instances were terminated if an optimal solution was not found within the time limit. In all instances, the confidence level of the Conditional Value-at-Risk was chosen as $\alpha = 0.9$. Columns of table 6.1 show $|V|$, number of nodes in the graph, *den*, the expected density of the graph, $\mathcal{N}$, number of scenarios for each instance, $\omega_r(G)$, size of the risk-averse maximum clique, and also the computational time (in seconds) and obtained cost for RAMCQ and CPLEX. It was observed that RAMCQ performs up to 10 times better than CPLEX for sparse graphs (density 25%). However, as the density of the graphs increases, the performance of RAMCQ compared to CPELX declines. This domination persists for graphs up to density 70%, and at that point CPLEX starts to outperform RAMCQ. The deterioration in RAMCQ's performance can be explained by the fact that the mixed integer formulation, solved by CPLEX, contains a constraint for each non-edge in the graph, which leads to the problem having more constrains for sparser graphs. However, as the density of the graph increases, the relative number of constrains decreases, and so does the difficulty of the mixed integer problem. It is interesting to note that for

problems with equal density and $|\mathcal{N}|$, the expected risk of the problem decreases as the number of nodes in the graph increases.

Table 6.1: Average optimal clique sizes and computation times in seconds obtained by RAMCQ and CPLEX

$\|V\|$	density	$\|\mathcal{N}\|$	$\omega_r(G)$	RAMCQ	CPLEX12.2
50	0.25	100	4.455	**0.174**	0.818
75	0.25	100	4.556	**0.514**	6.639
100	0.25	100	4.8	**0.913**	16.883
150	0.25	100	5.1	**2.883**	76.351
200	0.25	100	5	**7.672**	345.13
50	0.5	100	6.333	**0.721**	1.558
75	0.5	100	6.333	**3.117**	12.053
100	0.5	100	7	**7.638**	30.411
150	0.5	100	8.333	**55.201**	284.227
200	0.5	100	-	-	-
50	0.55	100	7.667	**1.015**	3.211
75	0.55	100	7.333	**5.058**	13.198
100	0.55	100	8.667	**17.391**	42.105
150	0.55	100	9.667	**116.082**	418.73
200	0.55	100	-	-	-
50	0.6	100	7	**1.97**	3.849
75	0.6	100	8	**9.646**	17.689
100	0.6	100	9.333	**39.943**	91.271
150	0.6	100	10.667	**190.472**	404.66
200	0.6	100	-	-	-
50	0.65	100	8.2	**2.597**	3.803
75	0.65	100	9.2	**13.528**	16.123
100	0.65	100	10.2	**62.467**	73.594
150	0.65	100	10.75	**407.933**	687.985
200	0.65	100	-	-	-
50	0.7	100	10	**3.662**	4.898
75	0.7	100	10	31.474	**23.364**
100	0.7	100	13	171.232	**148.066**
150	0.7	100	14	**503.358**	604.862
200	0.7	100	-	-	-

CHAPTER 7
CONCLUSIONS

In chapter 2, randomized multidimensional assignment problems that correspond to hypergraph matching problems were studied. Two different methods were provided to obtain guaranteed high-quality solutions for MAPs with linear sum or linear bottleneck cost functions, specifically designed for instances with small dimensionality or small cardinality. The computational results demonstrated that the proposed methods provide a tight upper bound on the value of the optimal cost for MAPs in randomized problems. The heuristic provided for problems with fixed cardinality can provide high-quality solutions to problems with large dimensionality in a relatively short time. The limiting factor for the heuristic method is the memory consumption. The structure of the proposed methods makes them suitable for parallel computing. As an extension, the performance of the proposed heuristic in a parallel system can be studied.

In chapter 3, bitset-based data structures were proposed for the algorithm presented by Grunert et al [30] for the problem of enumerating all k-cliques in a k-partite graph. Utilization of bitsets and the associated bit parallelism enables one to reduce the computational cost of branching and backtracking in the branch-and-bound procedure. Numerical experiments on small- and large-scale randomly generated k-partite graphs show that the proposed approach allows for achieving substantial computational improvements over the original method of [30].

The multi-dimensional assignment problem (MAP) were studied in chapter 4.

We proposed two heuristic methods to obtain approximate solutions for MAP, and showed how these solutions can be used in an exact method, such as a branch and bound method.

Some of the recent maximum clique branch and bound algorithms were reviewed in chapter 5. A new coloring method was proposed based on the unit propagation technique which is widely used in MaxSAT solvers. The new coloring method will work on top of graph coloring heuristics to obtain a smaller color number for the nodes. In the new coloring method, the colors assigned to the nodes is not necessarily the same as the index of the color class they are assigned to in the graph coloring heuristics. The new coloring method was used in a new algorithm for the maximum clique problem and the numerical results were provided.

In chapter 6, we studied the minimum-risk maximum clique problem, i.e. a risk-averse maximum clique problem on stochastic graphs. A new graph-theoretic method was proposed for this problem and used to solve randomly generated Erdos-Renyi with random factors associated to their nodes. Comparisons were made with CPLEX, and it was shown that the new method is very competitive.

REFERENCES

[1] S. Abdullah, E. K. Burke, and B. Mccollum. Using a randomised iterative improvement algorithm with composite neigjhbourhood structures for university course timetabling. In *in The Proceedings of the 6th Metaheuristic International Conference [MIC05*, pages 22–26. Book, 2005.

[2] YP Aneja, R Chandrasekaran, and KPK Nair. Maximizing residual flow under an arc destruction. *Networks*, 38(4):194–198, 2001.

[3] K. M. Anstreicher. Recent advances in the solution of quadratic assignment problems. *Mathematical Programming*, 97(1–2):27–42, 2003.

[4] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999.

[5] Alper Atamtürk and Muhong Zhang. Two-stage robust network flow and design under demand uncertainty. *Operations Research*, 55(4):662–673, 2007.

[6] E. Balas and P. R. Randweer. Traffic assignment in communications satellites. *Operations Research Letters*, 2(4):141–147, 1983.

[7] Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4):1054–1068, 1986.

[8] H. Bekker, E. P. Braad, and Boris Goldengorin. Using bipartite and multidimensional matching to select the roots of a system of polynomial equations. In *ICCSA (4)*, pages 397–406, 2005.

[9] Vladimir L Boginski, Clayton W Commander, and Timofey Turko. Polynomial-time identification of robust network flows under uncertain arc failures. *Optimization Letters*, 3(3):461–473, 2009.

[10] Immanuel M. Bomze, Marco Budinich, Panos M. Pardalos, and Marcello Pelillo. The maximum clique problem. In *Handbook of Combinatorial Optimization*, pages 1–74. Kluwer Academic Publishers, 1999.

[11] Immanuel M. Bomze, Marco Budinich, Marcello Pelillo, and Claudio Rossi. Annealed replication: a new heuristic for the maximum clique problem. *Discrete Appl. Math.*, 121(1-3):27–49, September 2002.

[12] W. L. Brogan. Algorithm for ranked assignments with applications to multiobject tracking. *Journal of Guidance, Control, and Dynamics*, 12(3):357–364, 1989.

[13] R. E. Burkard. Quadratic assignment problems. *European Journal of Operational Research*, 15(3):283–289, 1984.

[14] R. E. Burkard. Time-slot assignment for tdma systems. *Computing*, 35(2):99–112, 1985.

[15] R. E. Burkard. Selected topics on assignment problems. *Discrete Applied Mathematics*, 123(1–3):257–302, 2002.

[16] R. E. Burkard and E. Çela. Linear assignment problems and extensions. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization, Supplement Volume A*, pages 75–149. Kluwer Academic Publishers, Dordrecht, 1999.

[17] P. Carraresi and G. Gallo. A multi-level bottleneck assignment approach to the bus drivers' rostering problem. *European Journal of Operational Research*, 16(2):163–173, 1984.

[18] MW Carter and G Laporte. Recent developments in practical course timetabling. In Burke, E and Carter, M, editor, *Practice And Theory Of Automated Timetabling Ii*, volume 1408 of *Lecture Notes In Computer Science*, pages 3–19, Heidelberger Platz 3, D-14197 Berlin, Germany, 1998. Springer-Verlag Berlin. 2nd International Conference on the Practice and Theory of Automated Timetabling (Patat 97), Toronto, Canada, Aug 20-22, 1997.

[19] L Cavique, C Rego, and Isabel Themido. A scatter search algorithm for the maximum clique problem. *Instituto Politecnico de*, 48:0–16, 2001.

[20] D. Coppersmith and G. Sorkin. Constructive bounds and exact expectations for the random assignment problem. *Random Structures and Algorithms*, 15(2):113–144, 1999.

[21] W. E. Donath. Algorithm and average-value bounds for assignment problems. *IBM Journal of Research and Development*, pages 380–386, 1969.

[22] A. Dutta and P. Tsiotras. A greedy random adaptive search procedure for optimal scheduling of p2p satellite refueling. In *AAS/AIAA Space Flight Mechanics Meeting*, pages 07–150, 2007.

[23] Paul Erdös and A Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.

[24] Serge Fenet and Christine Solnon. Searching for maximum cliques with ant colony optimization. In *Proceedings of the 2003 international conference on Applications of evolutionary computing*, EvoWorkshops'03, pages 236–245, Berlin, Heidelberg, 2003. Springer-Verlag.

[25] R. Fulkerson, I. Glickberg, and O. Gross. A production line assignment problem. Technical Report RM-1102, The RAND Corporation, Sta. Monica, CA, 1953.

[26] Gregory D Glockner and George L Nemhauser. A dynamic network flow problem with uncertain arc capacities: formulation and problem structure. *Operations Research*, 48(2):233–242, 2000.

[27] S. Grabowski and K. Fredriksson. Bit-parallel string matching under hamming distance in $O(n\lceil m/w\rceil)O(n\lceil m/w\rceil)$ worst case time. *Information Processing Letters*, 105(5):182–187, 2008.

[28] M. Grötschel, L. Lovász, and A. Schrijver. Relaxations of vertex packing. *Journal of Combinatorial Theory, Series B*, 40(3):330–343, 1986.

[29] D. Grundel, P. Krokhmal, C. Oliveira, and P. Pardalos. On the number of local minima in the multidimensional assignment problem. *Journal of Combinatorial Optimization*, 13(1):1–18, 2007.

[30] T. Grünert, S. Irnich, H. Zimmermann, M. Schneider, and B. Wulfhorst. Finding all k-cliques in k-partite graphs, an application in textile engineering. *Computers & Operations Research*, 29(1):13–31, January 2002.

[31] H. Hyyrö. Bit-parallel approximate string matching algorithms with transposition. *Journal of Discrete Algorithms*, 3(2–4):215–229, 2005.

[32] H. Hyyrö and G. Navarro. Bit-parallel witnesses and their applications to approximate string matching. *Algorithmica*, 41(3):203–231, 2004.

[33] David S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, STOC '73, pages 38–49, New York, NY, USA, 1973. ACM.

[34] R. M. Karp. An upper bound on the expected cost of an optimal assignment. In D. S. Johnson, T. Nishizeki, A. Nozaki, and H. S. Wilf, editors, *Discrete Algorithms and Complexity*, volume 15 of *Perspectives in Computing*, pages 1–4. Academic Press, Boston, 1987.

[35] Kengo Katayama, Akihiro Hamamoto, and Hiroyuki Narihisa. Solving the maximum clique problem by k-opt local search. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 1021–1025, New York, NY, USA, 2004. ACM.

[36] Janez Konc and Dušanka Janezic. An improved branch and bound algorithm for the maximum clique problem. *proteins*, 4:5, 2007.

[37] T. C. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25(1):52–76, 1957.

[38] P. Krokhmal, D. Grundel, and P. Pardalos. Asymptotic behavior of the expected optimal value of the multidimensional assignment problem. *Mathematical Programming*, 109(2–3):525–551, 2007.

[39] P. A. Krokhmal and P. M. Pardalos. Limiting optimal values and convergence rates in some combinatorial optimization problems on hypergraph matchings. *Submitted for publication*, 2010.

[40] P. A. Krokhmal and P. M. Pardalos. Limiting optimal values and convergence rates in some combinatorial optimization problems on hypergraph matchings. *Submitted for publication*, 2011.

[41] PAVLO A Krokhmal. Higher moment coherent risk measures. 2007.

[42] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–87, 1955.

[43] J. M. Kurtzberg. On approximation methods for the assignment problem. *Journal of the ACM*, 9(4):419–439, 1962.

[44] A. J. Lazarus. Certain expected values in the random assignment problem. Technical report, Math. & Computer Science Dept., Univ of California, Riverside, 1990.

[45] Charles E. Leiserson, Harald Prokop, and Keith H. Randall. Using de Bruijn sequences to index a 1 in a computer word. *Working paper*, 1998, http://supertech.csail.mit.edu/papers/debruijn.ps.

[46] Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *AAAI*, 2010.

[47] Q Liu and YPP Chen. High functional coherence in k-partite protein cliques of protein interaction networks. *Bioinformatics and Biomedicine*, 2009.

[48] E. A. Loiola, N. M. Maia de Abreu, P. O. Boaventura-Netto, P. M. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007.

[49] M. Mirghorbani, P. Krokhmal, and E. L. Pasiliao. Computational studies of randomized multidimensional assignment problems. In Alexey Sorokin, My T. Thai, and Panos M. Pardalos, editors, *Dynamics of Information Systems*, page in press. Springer.

[50] G. L. Nemhauser and L. E. Trotter. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6:48–61, 1974. 10.1007/BF01580222.

[51] G. L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975. 10.1007/BF01580444.

[52] B. Olin. *Asymptotic properties of the random assignment problem*. PhD thesis, Department of Mathematics, Royal Institute of Technology, Stockholm, Sweden, 1992.

[53] Patric R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Appl. Math.*, 120(1-3):197–207, August 2002.

[54] P. M. Pardalos and H. Wolkowicz, editors. *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, 1994.

[55] D. W. Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793, 2007.

[56] M Peters. CLICK: Clustering categorical data using k-partite maximal cliques. *IEEE International Conference on Data Engineering*, 2005.

[57] W. Pierskalla. The multidimensional assignment problem. *Operations Research*, 16(2):422–431, 1968.

[58] A. B. Poore. Multidimensional assignment formulation of data association problems arising from multitarget and multisensor tracking. *Computation Optimization and Applications*, 3(1):27–54, 1994.

[59] Wayne Pullan and Holger H. Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, 2006.

[60] JF Pusztaszeri, PE Rensing, and TM Liebling. Tracking elementary particles near their primary vertex: A combinatorial approach. *Journal Of Global Optimization*, 9(1):41–64, JUL 1996. 3rd Workshop on Global Optimization, SZEGED, HUNGARY, DEC, 1995.

[61] R Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

[62] R Tyrrell Rockafellar and Stanislav Uryasev. Conditional value-at-risk for general loss distributions. *Journal of Banking & Finance*, 26(7):1443–1471, 2002.

[63] M. Rysz, P. Krokhmal, and E. L. Pasiliao. Minimum risk maximum clique problem. In Alexey Sorokin and Panos M. Pardalos, editors, *Dynamics of Information Systems*, page in press. Springer.

[64] T. Sahni and T. Gonzales. P-complete approximation problems. *Journal of the Association for Computing Machinery*, 23(3):555–565, 1976.

[65] Pablo San Segundo, Diego Rodríguez-Losada, and Agustín Jiminez. An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38(2):571–581, February 2011.

[66] Pablo San Segundo, Cristbal Tapia, Julio Puente, and Diego Rodrguez-Losada. A new exact bit-parallel algorithm for sat. In *ICTAI (2)'08*, pages 59–65, 2008.

[67] Alexey Sorokin, Vladimir Boginski, Artyom Nahapetyan, and Panos M Pardalos. Computational risk management techniques for fixed charge network flow problems with uncertain arc failures. *Journal of Combinatorial Optimization*, 25(1):99–122, 2013.

[68] L. Steinberg. The backboard wiring problem: a placement algorithm. *SIAM Review*, 3(1):37–50, 1961.

[69] Etsuji Tomita and Tomokazu Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In *DMTCS'03: Proceedings of the 4th international conference on Discrete mathematics and theoretical computer science*, pages 278–289, Berlin, Heidelberg, 2003. Springer-Verlag.

[70] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *Proceedings of the 4th International Workshop WALCOM 2010*, volume 5942 of *Lecture Notes in Computer Science*, pages 191–203. Springer, 2010.

[71] Satish V Ukkusuri, Tom V Mathew, and S Travis Waller. Robust transportation network design under demand uncertainty. *Computer-Aided Civil and Infrastructure Engineering*, 22(1):6–18, 2007.

[72] TL Urban and RA Russell. Scheduling Sports Competitions On Multiple Venues. *European Journal Of Operational Research*, 148(2):302–311, JUL 16 2003.

[73] Bram Verweij, Shabbir Ahmed, Anton J Kleywegt, George Nemhauser, and Alexander Shapiro. The sample average approximation method applied to stochastic routing problems: a computational study. *Computational Optimization and Applications*, 24(2-3):289–333, 2003.

[74] D. W. Walkup. On the expected value of a random assignment problem. *SIAM Journal on Computation*, 8(3):440–442, 1979.