FACETS OF CONFLICT HYPERGRAPHS

A Thesis
Presented to
The Academic Faculty

by

Siddhartha Maheshwary

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
December 2008

FACETS OF CONFLICT HYPERGRAPHS

Dedicated to that mysterious Power:

Who requires odd holes for imperfection in graphs

And ensures unimodularity of network flows

Who converges large numbers towards normality

And lets greediness work for the spanning tree

Who animates the intelligence inside us

And creates the world-illusion around us

She is the seed of all creativity

And without Her grace

This work would forever remain

a mere potentiality

ACKNOWLEDGEMENTS

I would like to thank the following persons

for their advice and support:

Dr. Eva K. Lee

Dr. Ellis Johnson

Dr. Earl Barnes

Dr. R. Gary Parker

Dr. D. J. Wu

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

We study the facial structure of the independent set polytope using the concept of conflict hypergraphs. A conflict hypergraph is a hypergraph whose vertices correspond to the binary variables, and edges correspond to covers of the constraint matrix of the independent set polytope. Various structures such as cliques, odd holes, odd anti-holes, webs and anti-webs are identified on the conflict hypergraph. These hypergraph structures are shown to be generalization of traditional graph structures. Valid inequalities are derived from these hypergraph structures, and the facet defining conditions are studied. Chvatal-Gomory ranks are derived for odd hole and clique inequalities. To test the hypergraph cuts, we conduct computational experiments on *market-share* (also referred to as *market-split*) problems. These instances consist of 100% dense multiple-knapsack constraints. They are small in size but are extremely hard to solve by traditional means. Their difficult nature is attributed mainly to the dense and symmetrical structure. We employ a special branching strategy in combination with the hypergraph inequalities to solve many of the particularly difficult instances. Results are reported for serial as well as parallel implementations.

CHAPTER I

INTRODUCTION

We begin by defining an *independence system.* Let $N = \{1, \ldots, n\}$ be a set of base elements. Let $\mathcal{I}$ be a collection of subsets of N such that $\emptyset \in \mathcal{I}$, and $I_1 \subset I_2 \in \mathcal{I}$ implies $I_1 \in \mathcal{I}$. Then the pair $S = (N, \mathcal{I})$ is called an independence system. Members of $\mathcal{I}$ are called *independent sets.* Any $H \subset N$ such that $H \notin \mathcal{I}$ is called a *dependent set.* The combinatorial optimization problem associated with S is the following: Given a weight w_j for each $j \in N$, find the independent set that has the maximum possible total weight. This combinatorial optimization problem is referred to as the *independent set problem.*

The independent set problem occurs in many areas of operations research. Some of the well known problems in graphs theory (such as vertex packing and maximum clique problem) can be shown to be equivalent to the independent set problem. Knapsack problem is another well-known problem that can be modeled as an independent set problem. Many variants of packing and covering problems are also closely related to it. The reader may refer to [35, 38] for a detailed exposition.

In this thesis we are concerned with the following form of the independent set problem: $ISP = \{x \in \{0,1\}^n : Ax \leq b\}$, where $A \in \mathbb{R}_+^{m \times n}$, and $b \in \mathbb{R}_+^m$ (m and n being positive integers).

Many of the successful strategies for solving ISP involve generation of strong cutting planes. Among all the cutting planes, the strongest are those that define facets of the independent set polytope $P^{ISP} = conv(ISP)$, which is the convex hull of feasible solutions to ISP [17, 26, 32, 35, 27].

Research concerning the facial structure of P^{ISP} has been reported since the early

1970's. In this thesis, we investigate the facial structure of P^{ISP} via the concept of conflict hypergraphs.

Padberg [36] introduced the notion of intersection graphs in context of set packing problems. He showed that various graph structures such cliques and odd holes on the intersection graph can be used to derive facets of the set packing polyhedra. Another early paper on covering, packing and knapsack problems is by Padberg [37]. Cornuéjols and Sassano [15] studied the facial structure of the set covering polytope by defining a bipartite incidence graph, and using critical edges and cutsets on that graph to derive facets.

Several researchers have extended these ideas. Golumbic talked about interval graphs [24]. Others used *conflict graphs* that can be defined on any constraint matrix (not just $0-1$ matrices found in set covering and packing). The vertices of such a conflict graph correspond to the binary variables, and two vertices are connected by an edge if the associated variables cannot both be equal to 1. This concept was utilized by Lee and Bixby [30, 10] to solve truck dispatching and scheduling problems. Atamtürk et al. [5] developed algorithms and data structures for effective and efficient construction, management, and utilization of dynamically changing conflict graphs. Atamtürk et al. [6] also studied the mixed vertex packing problem. Borndörfer [12] studied set packing, covering and partitioning problems.

A number of researchers have tried to generalize the notion of graph structures such as cliques and cycles. Euler et al. [21] generalized cliques, odd cycles and anticycles to get facets of independence system polyhedra. Laurent [29] generalized antiwebs. Müller and Schulz [33] presented the idea of *transitive packings.* Another important paper is by Sekiguchi [40]. He investigated the node packing problem on hypergraphs, and mentioned the connection between the knapsack polytope and hypergraphs. Easton et al. [18] extended the idea of conflict graphs, and used conflict hypergraphs to generate facets for P^{ISP}. They identified hypercliques in the conflict hypergraph, and derive facet-defining inequalities from them.

In this thesis, we identify structures such as odd holes, odd antiholes, cliques, webs and antiwebs in the conflict hypergraph. We utilize these structures to obtain valid inequalities for P^{ISP}, and identify conditions under which these valid inequalities become facet-defining. We also investigate the Chvatal-Gomory ranks of these inequalities, and investigate the separation problem. Computational experiments will be presented to verify the usefulness of conflict hypergraphs. Results from serial as well as parallel implementations are presented.

CHAPTER II

THEORETICAL INVESTIGATION

In this chapter, we present the theoretical results pertaining to conflict hypergraphs, valid inequalities and facets. We begin by describing the notation and summarizing some of the preliminary fundamental concepts. We then provide definitions of various hypergraph structures, and derive theoretical results related to these structures. We derive valid inequalities, and investigate the facet-defining conditions. Chvatal-Gomory ranks are studied for some of the valid inequalities. Finally, we investigate the issue of separation.

2.1 Preliminaries

The concepts of graphs and hypergraphs are used extensively in this work. Berge [8] is a good reference for both subjects. Formally, a graph G consists of a finite set of vertices $V(G) = \{1, \ldots, n\}$ and a set of edges $E(G) = \{e_1, \ldots, e_q\}$ where elements of $E(G)$ are subsets of $V(G)$ of size 2. A hypergraph $H = (V(H), E(H))$ consists of a set of vertices $V(H)$ and a set of edges $E(H) \subseteq 2^{V(H)}$ where $2^{V(H)}$ denotes the power set of $V(H)$. Hypergraphs with a constant edge size are also known as *uniform hypergraphs*. A uniform k-hypergraph is a hypergraph in which the cardinality of each edge is k. A graph is equivalent to a uniform 2-hypergraph. Figure 1 visually depicts a hypergraph H with vertex set $V(H) = \{1, \ldots, 21\}$ and edge set $E(S) = \{e_1, \ldots, e_6\}$.

A hypergraph $S = (V(S), E(S))$ is called an *induced sub-hypergraph* of a hypergraph H if $E(S) \subseteq E(H)$. In Figure 1, hypergraph S with vertex set $V(S) = \{1, 2, 3, 4, 19, 20, 21\}$ and edge set $E(S) = \{e_1, e_2\}$ can be considered an induced sub-hypergraph in H.

A *vertex packing* $I \subseteq V(H)$ on a hypergraph H is defined as a set of vertices for

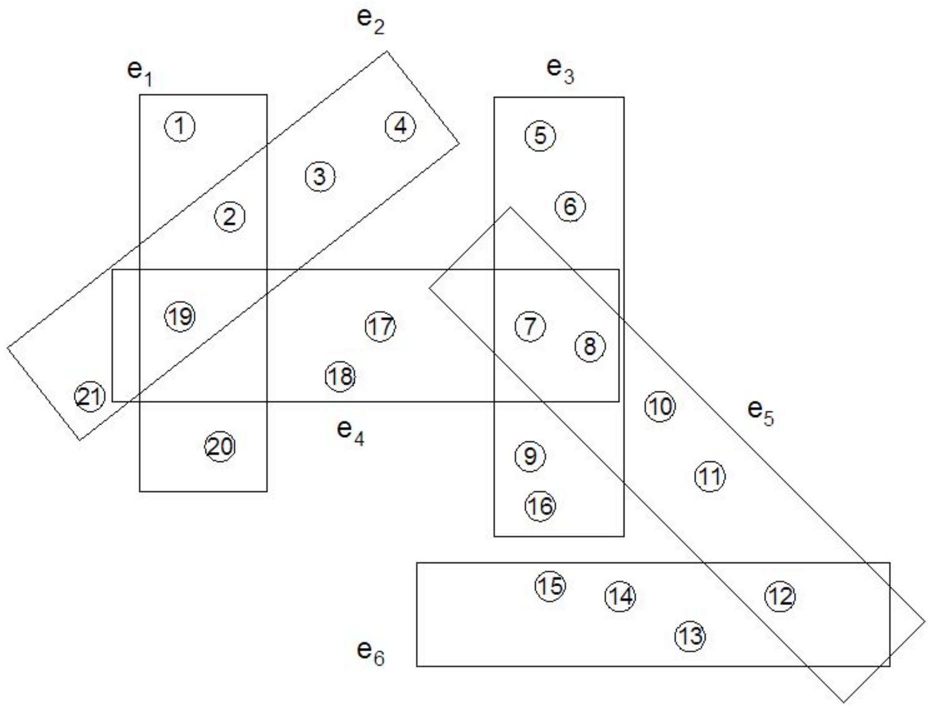


Figure 1: Example of a hypergraph.

which there does not exist an edge $e \in E(H)$ such that $e \subseteq I$. Thus, a vertex packing cannot contain all the vertices of an edge. A vertex packing on a hypergraph will be called *maximum* if there does not exist another packing with a higher cardinality. For brevity, we will sometimes refer to vertex packing as simply a packing.

Refer again to Figure 1. The set $I_1 = \{5, 6, 7, 8\}$ is a vertex packing. So is the set $I_2 = \{2, 3, 19, 21, 20, 17, 18\}$. Neither of these packings are maximum. The set $\{1, 2, 19, 20\}$ is not a valid packing because all the vertices of the edge e_1 are selected.

For a hypergraph H, we denote by u_j the unit vector of length $|V(H)|$ with a 1 in j^{th} place and 0's elsewhere. Thus, u_j is the characteristic vector to denote that the vertex $j \in V(H)$ is selected. Also, let $u_P = \sum_{j \in P} u_j$ for any $P \subseteq V(H)$. Thus, u_P is the characteristic vector to denote that the subset P of vertices in the hypergraph H is chosen.

The dimension of polytope P is d, denoted by $dim(P) = d$, if the maximum number of affinely independent points in P is $d + 1$. An inequality $(\pi, \pi_0) \in \mathbb{R}^n \times \mathbb{R}$

is *valid* for a polytope P if $\pi x \leq \pi_0 \ \forall \ x \in P$. In this case, $\mathcal{F} = \{x \in P : \pi x = \pi_0\}$ is called a *face* of P. If the dimension of $\mathcal{F}$ is one less than the dimension of P, then $\mathcal{F}$ is called a *facet* of P and the valid inequality (π, π_0) is said to be *facet-defining* for P. The reader can refer to Nemhauser and Wolsey [35] for details.

Consider the independent set polytope P^{ISP} (defined earlier). Without loss of generality, we assume $a_{ij} \leq b_i$ for all $j = 1, \ldots, n$, $i = 1, \ldots, m$. Under these assumptions, the set of unit vectors $\{u_j : j = 1, \ldots, n\}$ along with the 0 vector are $n + 1$ affinely independent points that are feasible to P^{ISP}. Therefore, P^{ISP} is *full-dimensional*, that is, $dim(P^{ISP}) = n$.

Every ISP instance induces a *conflict hypergraph* H, where $V(H) = \{1, \ldots, n\}$, and $E(H) = \{e \subset V(H) : \sum_{j \in e} A_j \not\leq b\}$ where A_j indicates the j^{th} column of matrix A.

Thus, every variable of the ISP instance corresponds to a vertex of H, and a subset of vertices comprises an edge if and only if the corresponding variables in the ISP instance form a dependent set (cover). We will use the words vertex and variables interchangeably because of the one-to-one correspondence between them.

An edge is said to be *minimal* if it represents a minimal dependent set (or a minimal cover) in the ISP instance. A conflict hypergraph will be called *minimal* if all of its edges are minimal. The concept of a minimal conflict hypergraph will be used later to facilitate the derivation of certain theoretical results.

Consider the following ISP instance given in Example 2.1.1. The minimal conflict hypegraph induced by this instance is shown in figure 2. The hypergraph contains vertices $\{1, \ldots, 20\}$. There exists an edge for every minimal cover. For example, $\{x_1, x_{19}, x_{20}\}$ form a minimal cover in the second constraint of the ISP instance. Therefore, the hypergraph contains the edge $\{1, 19, 20\}$. Similarly, $\{x_{10}, x_{11}\}$ form a minimal cover in the sixth constraint of the ISP instance. Hence, the hypergraph contains the edge $\{10, 11\}$. Other edges are generated in a similar manner.

Example 2.1.1.

$$
\begin{aligned}
&2x_1 + 4x_2 + 3x_3 + x_4 + x_5 \leq 10\\
&x_1 + x_{19} + x_{20} \leq 2\\
&x_4 + x_5 + x_6 + x_7 \leq 3\\
&3x_{19} + 2x_{18} + 5x_{11} \leq 9\\
&2x_6 + 3x_7 + x_{11} \leq 5\\
&x_{10} + x_{11} \leq 1\\
&x_{10} + 2x_{12} + 3x_{17} + 4x_{19} \leq 9\\
&x_{10} + x_{13} \leq 1\\
&x_{13} + x_{16} \leq 1\\
&x_{13} + x_{14} + x_{15} \leq 2\\
&2x_6 + 5x_7 + 10x_8 + 7x_9 \leq 23\\
&x_9 + x_{11} \leq 1\\
&23x_6 + 13x_7 + 17x_{10} \leq 50\\
&29x_9 + 84x_{10} \leq 86\\
&x_j \in \{0,1\} \quad \forall j = 1, \ldots, 20
\end{aligned}
$$

The vertex packing problem on H is defined as $VPP = \{x \in \{0,1\}^n : \sum_{j \in e} x_j \leq |e| - 1, e \in E(H)\}$. Let $P^{VPP} = conv(VPP)$.

Observe that a feasible solution to the ISP instance represents a feasible vertex packing on the conflict hypergraph, but a feasible vertex packing on the conflict hypergraph may not represent a feasible solution to the ISP instance. However, a feasible vertex packing on a minimal conflict hypergraph represents a feasible solution to the ISP instance.

For an induced sub-hypergraph S of H, we define P_S^{ISP} to be P^{ISP} restricted only to variables in S; that is, $P_S^{ISP} = conv\{x \in \{0,1\}^n : Ax \leq b, x_j = 0 \; \forall j \notin V(S)\}$.

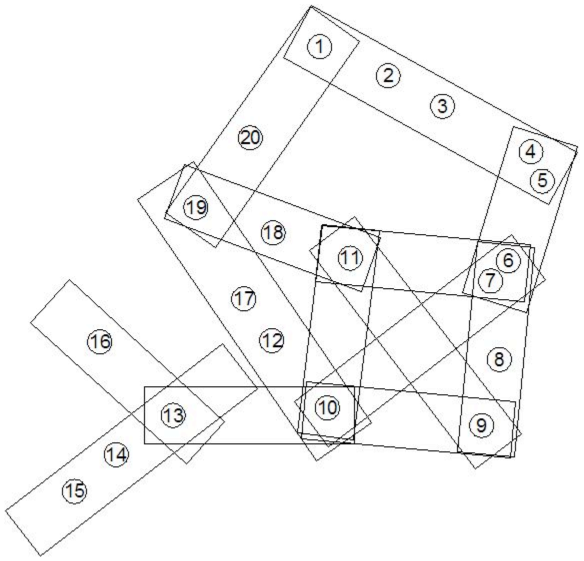


Figure 2: Conflict hypergraph for the ISP instance given in Example 2.1.1.

2.2 Hypergraph Structures

In this section we define several hypergraph structures that will be investigated in subsequent sections. A conflict hypergraph generated from an ISP instance can have a very broad range of structures. Herein, the definitions are somewhat restrictive, as they are defined to facilitate the proof of certain theoretical results for a large group of hypergraph structures. The definitions and results will be extended to more general cases in Section 2.5.

Definition. 2.2.1 For integer $d \geq 3$, let $P^1, \ldots, P^d$ be finite, mutually disjoint, non-empty sets. Let S be a hypergraph with vertex set $V(S) = \bigcup_{j=1}^{d} P^j$. We will refer to P^j's as *partitions* of S, $j = 1, \ldots, d$. Let k be an integer such that $1 \leq k \leq \frac{d}{2}$.

1. If d is odd and S has the edge set $E(S) = \{P^i \cup P^{i+1} : i = 1, \ldots, d, P^{d+1} \equiv P^1\}$, then S is called an *odd hole*.

2. If d is odd and S has the edge set $E(S) = \{P^i \cup P^j : |i-j| \geq 2, i, j = 1, \ldots, d\}$, then S is called an *odd antihole*.

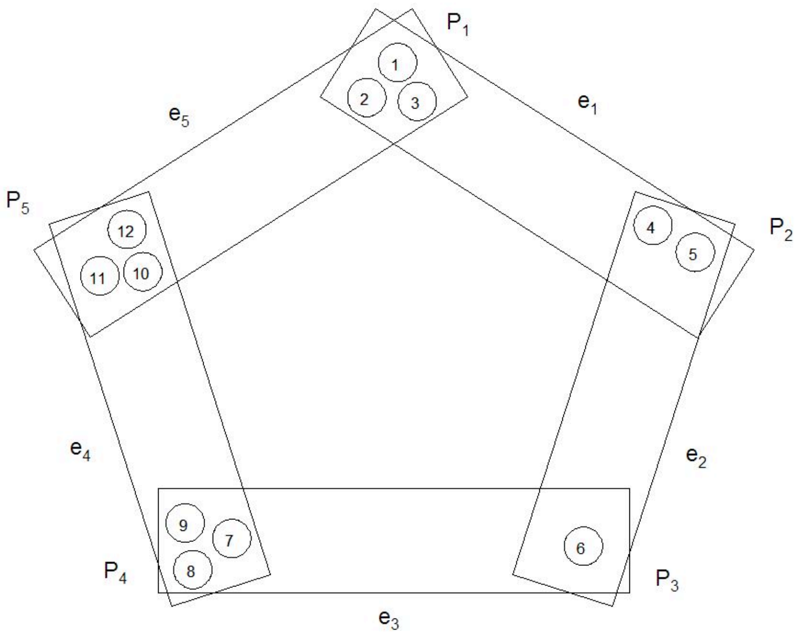


Figure 3: Odd hole with five partitions.

3. If S has the edge set $E(S) = \{P^i \cup P^j : i, j = 1, \ldots, d,\ i \neq j\}$, then S is called a *clique*.

4. If S has the edge set $E(S) = \{P^i \cup P^j : i = 1, \ldots, d, j = i + k, \ldots, i + d - k\}$, where $P^{d+l} \equiv P^l$ for all $l = 1, \ldots, d$, then S is called a *web*.

5. If S has the edge set $E(S) = \{P^i \cup P^j : i = 1, \ldots, d, j = 1, \ldots, i + k - 1, i + d - k + 1, \ldots, d\}$, where $P^{d+l} \equiv P^l$ for all $l = 1, \ldots, d$, then S is called an *antiweb*.

For the the last two definitions, we will call k the *step size* of the web or the antiweb.

In all these definitions, if all the partitions were singletons, then the cardinality of all edges becomes two, in which case they reduce to the corresponding graph structures.

Figures 3, 4, 5, 6 and 7 give examples of each of the defined hypergraph structures. The hypergraph shown in Figure 3 represents an odd hole consisting of five partitions. The edges and the partitions are labeled. For example, vertices $1, 2, 3$ comprise one

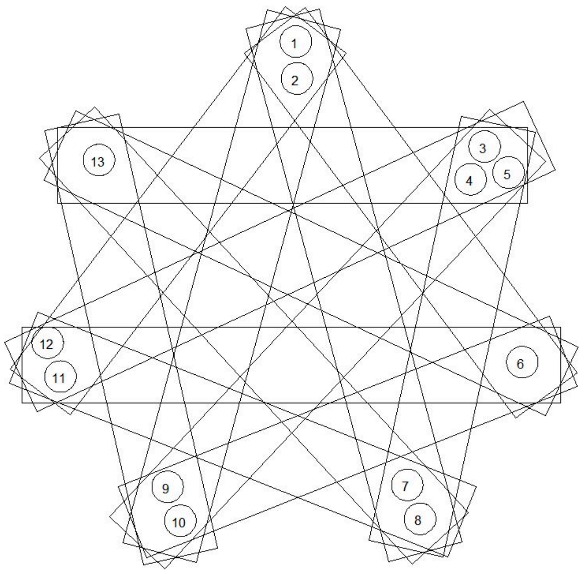


Figure 4: Odd antihole with seven partitions.

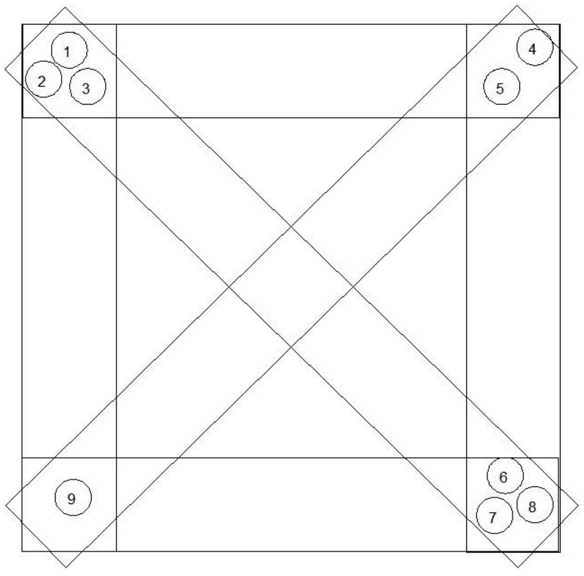


Figure 5: Clique with four partitions.

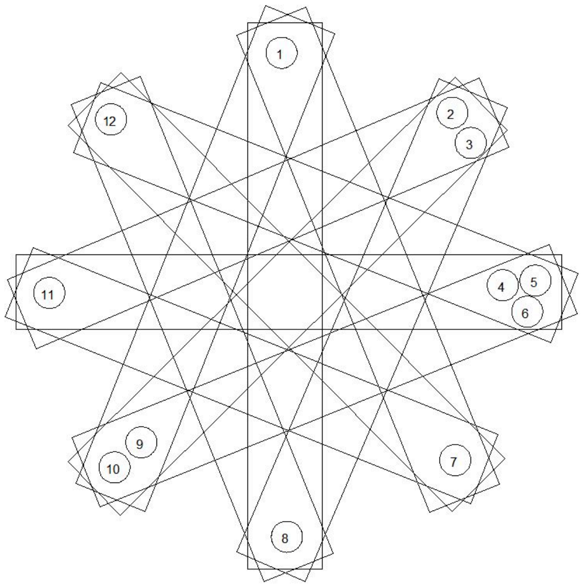


Figure 6: Web with eight partitions and step size three.

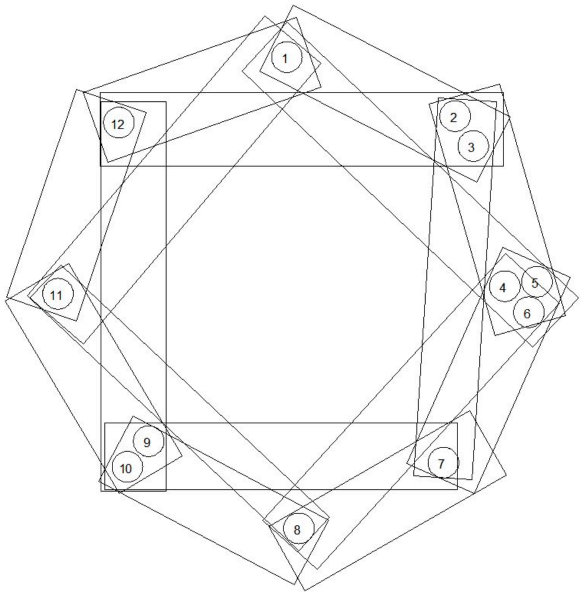
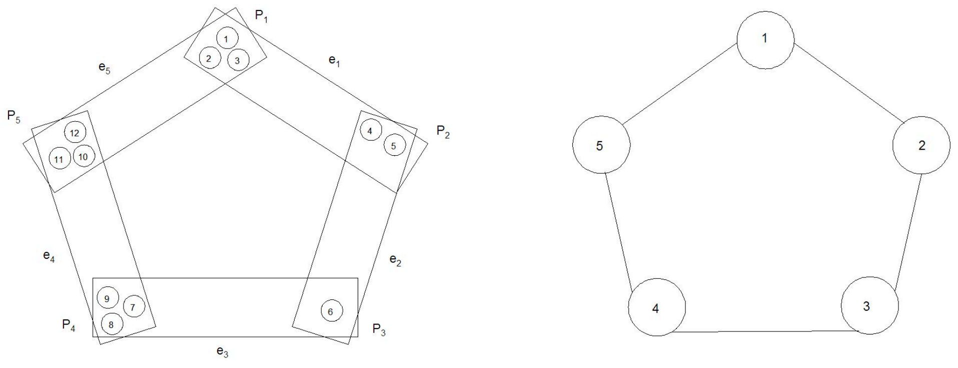


Figure 7: Antiweb with eight partitions and step size three.



(a) Hypergraph (b) Reduced graph

Figure 8: Odd hole with five partitions and the associated reduced graph.

partition, P_1. Similarly, the lone vertex 6 comprises another partition, namely, P_3. An edge is comprised by vertices in two partitions. For example, the edge e_5 is comprised by the vertices in partitions P_5 and P_1. Thus, $e_5 = \{P_5 \cup P_1\} = \{1, 2, 3, 10, 11, 12\}$.

Observe that the clique in Definition 2.2.1 is more relaxed and general than the *hyperclique* described by Easton et al. [18]. In [18], a hyperclique $K_{m,k}$ is defined as a uniform k-hypergraph with m vertices such that each of the $\binom{m}{k}$ combinations of vertices comprise an edge. Our definition of the clique does not insist on a uniform hypergraph, and does not requires as many edges.

For any structure S defined above, construct a graph G with vertex set $V(G) = \{1, \ldots, d\}$ and edge set $E(G) = \{\{i, j\} : P^i \cup P^j \in E(S)\}$. We will refer to G as the *reduced graph* of S. The reduced graph concept is illustrated in figure 8. As shown in the illustration, the partition P_j in the hypergraph corresponds to the vertex j in the reduced graph, $j = 1, \ldots, 5$.

2.3 Facets

We will now derive results for the various structures defined in Section 2.2. Note that Propositions 2.3.1 and 2.3.2 are for these structures as stand-alone hypergraphs and do not necessarily relate to conflict hypergraphs generated from ISP instances.

Proposition 2.3.1. *Let S be an odd hole, odd antihole, web, antiweb or clique. Let G*

be the associated reduced graph. Let $I(G) \subseteq V(G)$ be a maximum vertex packing on G. Then the cardinality of a maximum vertex packing on S is $|V(S)| - |V(G)| + |I(G)|$.

Proof. Observe that in any maximum vertex packing, a partition either contributes all of its vertices, or all-but-one of its vertices. To prove this, suppose there exists a maximum vertex packing in which a partition contributes at most all-but-two of its vertices. Now, each edge in S consists of two partitions. Therefore, we could always select one more vertex from that partition, since each edge can contribute all-but-one of its vertices. Thus, we can increase the cardinality of the original packing, which contradicts the assumption that we started with a maximum vertex packing.

Let $P^1, \ldots, P^d$ be the partitions of S. We claim that the set $I(S) = \{\bigcup_{j \in I(G)} P^j\} \cup \{\bigcup_{j \in V(G) \setminus I(G)} \bar{P}^j\}$ represents a maximum vertex packing on S, where $\bar{P}$ represents a set obtained by removing an element of P. Let $X = \{j : j \in V(G) \setminus I(G)\}$. Therefore, $|I(S)| = |V(S)| - |X|$.

To show that $I(S)$ indeed represents a maximum vertex packing on S, suppose there exists a maximum packing, say $I'(S)$, with a higher cardinality than that of $I(S)$. This implies that every partition contributes either all of its vertices, or all-but-one of its vertices to $I'(S)$. Let Y be the index set of the partitions that contribute all-but-one of their vertices, respectively. Therefore, $|I'(S)| = |V(S)| - |Y|$. Now, by assumption, $|I'(S)| > |I(S)|$. Therefore, $|X| > |Y|$. This shows, that a lesser number of partitions contribute all-but-one of their vertices to $I'(S)$. Hence, a greater number of partitions contribute all of their vertices in $I'(S)$.

Let Z be the index set of all the partitions that contribute all of their vertices in $I'(S)$. Now, because of one-to-one correspondence between vertices of G and partitions of S, Z should represent a vertex packing on G (because no two partitions that contribute all of their vertices can be connected by an edge). Also, $I(G)$ represents the index set of partitions contributing all of their vertices to $I(S)$. Therefore

$|Z| > |I(G)|$. But this contradicts the original assumption about $I(G)$ being a maximum vertex packing on G. This contradiction proves that there cannot exist a vertex packing that has a higher cardinality than $I(S)$. In other words, $I(S)$ is a maximum vertex packing.

It now remains to compute the cardinality of $I(S)$.

$$\begin{aligned} |I(S)| &= \sum_{j \in I(G)} |P^j| + \sum_{j \in V(G) \setminus I(G)} (|P^j| - 1) \\ &= \sum_{j \in V(G)} |P^j| - \sum_{j \in V(G) \setminus I(G)} (1) \\ &= |V(S)| - (|V(G)| - |I(G)|) \end{aligned}$$

$\square$

We remark that vertex packings on graph structures have been well-studied in cases related to the independent set problem. Padberg [36] has investigated cliques, odd holes and odd antiholes. Trotter [41] gives a detailed exposition on webs and antiwebs, and shows that these are a generalized form of odd holes, antiholes and cliques.

The next proposition enumerates maximum vertex packings with affinely independent characteristic vectors. This result will be used in proofs related to facets of P^{ISP}.

Proposition 2.3.2. *Let S be an odd hole, odd antihole, web, antiweb or clique. Let the step size k for the web or antiweb be such that k and d are relatively prime. Then, there exist $|V(S)|$ maximum vertex packings on S with affinely independent characteristic vectors.*

Proof. Let $P^1, \ldots, P^d$ be the partitions of S. For $j = 1, \ldots, d$, let the elements of P^j be indexed such that P^j_i indicates the i^{th} element, where $1 \leq i \leq |P^j|$. The vertex set $V(S) = \bigcup_{j=1}^{d} P^j$.

Let G be the reduced graph derived from S. Recall that there exists a one-to-one correspondence between the vertices of G and partitions of S. Let $V(G) = \{1, \ldots, d\}$.

We will prove the proposition by constructing the requisite vertex packings on S. We will perform this construction by using maximum vertex packings on G.

Let x^1 be a characteristic vector of a maximum vertex packing in G such that $x_1^1 = 0$. Construct x^i such that $x_{j+1}^{i+1} = x_j^i$, $i = 1, \ldots, d$, $j = 1, \ldots, d$, where index $d + 1 \equiv 1$. Therefore, x^i, $i = 1, \ldots, d$ are affinely independent vectors representing maximum vertex packings on G, and $x_i^i = 0$ for all $i = 1, \ldots, d$. This is true for odd holes, odd antiholes and cliques. Because k and d are relatively prime, this also holds for webs and antiwebs [41].

Now, for a given $i \in \{1, \ldots, d\}$, let $Y^i \in \{0, 1\}^{|V(S)|}$ such that

$$Y^i = \{\sum_{x_j^i=1} u_{P^j} + \sum_{x_j^i=0, j\neq i} u_{P^j \setminus \{P_1^j\}} + u_{P^i \setminus \{P_l^i\}} : l = 1, \ldots, |P^i|\}$$

The collection $\{Y^1, \ldots, Y^d\}$ contains a total of $\sum_{i=1}^d |P^i| = |V(S)|$ characteristic vectors. Moreover, each of these vectors represents a maximum vertex packing on S, and all the vectors are affinely independent. $\square$

We will now illustrate the construction of these vectors using an odd hole with d partitions as an example. The following matrix illustrates the maximum vertex packing on the reduced graph (whose vertices are indexed 1 through d).

$$\begin{array}{cc} & \begin{array}{ccccccc} x^1 & x^2 & x^3 & \cdots & x^{d-2} & x^{d-1} & x^d \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ d-2 \\ d-1 \\ d \end{array} & \left(\begin{array}{ccccccc} 0 & 0 & 1 & \cdots & 1 & 0 & 1 \\ 1 & 0 & 0 & \ddots & 0 & 1 & 0 \\ 0 & 1 & 0 & \ddots & 1 & 0 & 1 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 1 & 0 & \ddots & 0 & 0 & 1 \\ 1 & 0 & 1 & \ddots & 1 & 1 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & 0 \end{array}\right) \end{array}$$

Using the above matrix, we can construct the characteristic vectors for the odd hole. First, we need to define the following matrices. Let $\mathbf{1}_{m,n}$ be a matrix with m rows and n columns consisting of all 1's. Let $\overline{I}_m = \mathbf{1}_{m,m} - I_m$, where I_m is the identity matrix. Finally, let $A_{m,n} = (a_{i,j})_{i=1,\ldots,m;j=1,\ldots,n}$ be a matrix such that

$$a_{i,j} = \begin{cases} 0 & \text{if } i = 1; \\ 1 & \text{otherwise} \end{cases}$$

Thus, $A_{m,n}$ is a matrix with 0's in the top row, and 1's elsewhere.

Now consider the following matrix:

$$\begin{array}{cc} & \begin{array}{ccccccc} Y^1 & Y^2 & Y^3 & \cdots & Y^{d-2} & Y^{d-1} & Y^d \end{array} \\ \begin{array}{c} P^1 \\ P^2 \\ P^3 \\ \vdots \\ P^{d-2} \\ P^{d-1} \\ P^d \end{array} & \left(\begin{array}{ccccccc} \overline{I}_{|P^1|} & A_{|P^1|,|P^2|} & \mathbf{1}_{|P^1|,|P^3|} & \cdots & \mathbf{1}_{|P^1|,|P^{d-2}|} & A_{|P^1|,|P^{d-1}|} & \mathbf{1}_{|P^1|,|P^d|} \\ \mathbf{1}_{|P^2|,|P^1|} & \overline{I}_{|P^2|} & A_{|P^2|,|P^3|} & \ddots & A_{|P^2|,|P^{d-2}|} & \mathbf{1}_{|P^2|,|P^{d-1}|} & A_{|P^2|,|p^d|} \\ A_{|P^3|,|P^1|} & \mathbf{1}_{|P^3|,|P^2|} & \overline{I}_{|P^3|} & \ddots & \mathbf{1}_{|P^3|,|P^{d-2}|} & A_{|P^3|,|P^{d-1}|} & \mathbf{1}_{|P^3|,|P^d|} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ A_{|P^{d-2}|,|P^1|} & \mathbf{1}_{|P^{d-2}|,|P^2|} & A_{|P^{d-2}|,|P^3|} & \ddots & \overline{I}_{|P^{d-2}|} & A_{|P^{d-2}|,|P^{d-1}|} & \mathbf{1}_{|P^{d-2}|,|P^d|} \\ \mathbf{1}_{|P^{d-1}|,|P^1|} & A_{|P^{d-1}|,|P^2|} & \mathbf{1}_{|P^{d-1}|,|P^3|} & \ddots & \mathbf{1}_{|P^{d-1}|,|P^{d-2}|} & \overline{I}_{|P^{d-1}|} & A_{|P^{d-1}|,|P^d|} \\ A_{|P^d|,|P^1|} & \mathbf{1}_{|P^d|,|P^2|} & A_{|P^d|,|P^3|} & \cdots & A_{|P^d|,|P^{d-2}|} & \mathbf{1}_{|P^d|,|P^{d-1}|} & \overline{I}_{|P^d|} \end{array}\right) \end{array}$$

Each column of this matrix represents a maximum vertex packing on the odd hole. All these vectors are in fact linearly independent.

The next theorem derives the valid inequality associated with a structure (odd hole, odd antihole, web, antiweb or clique) existing as an induced sub-hypergraph in a conflict hypergraph.

Theorem 2.3.3. *Let H be a conflict hypergraph generated from an ISP instance. Let S be an odd hole, odd antihole, web, antiweb or clique existing as an induced sub-hypergraph in H. Let G be reduced graph associated with S. Let $I(G) \subset V(G)$ be a maximum vertex packing on G. Then the following is a valid inequality for P^{ISP}:*

$$\sum_{j \in V(S)} x_j \leq |V(S)| - |V(G)| + |I(G)| \tag{1}$$

Proof. Clearly, the number of vertices that can be selected from S can never exceed $|V(S)| - |V(G)| + |I(G)|$, because this is the cardinality of a maximum vertex packing on S (Proposition 2.3.1). Therefore, the inequality is valid. $\square$

Theorem 2.3.4. *Let H be a minimal conflict hypergraph induced by an ISP instance. Let S be an odd hole, odd antihole, web, antiweb or clique existing as an induced sub-hypergraph in H. Let the step size k of the web and antiweb be such that k and d are relatively prime. Then inequality (1) is facet-defining for P_S^{ISP}.*

Proof. Consider the unit vectors u_j, $j \in V(S)$, along with the 0 vector. These $|V(S)|+1$ vectors are affinely independent and feasible for P_S^{ISP}. Therefore, $dim(P_S^{ISP}) = |V(S)|$.

Therefore, we need to show that there exist $|V(S)|$ affinely independent points that satisfy (1) at equality. These points may be constructed from the packings described in Proposition 2.3.2 because all the edges of S are minimal. $\square$

Theorem 2.3.4 is similar to results derived for some of the graph structures in the past (see Padberg [36], Nemhauser and Trotter [34], and Trotter [41]) because it proves that the valid inequality is facet-defining for P_S^{ISP}, but not necessarily for P^{ISP}. We would need to lift the valid inequalities to get facets for P^{ISP}. A detailed exposition on lifting can be found in Nemhauser and Wolsey [35].

However, hypergraph structures can, under certain conditions, induce facets for P^{ISP}. This eliminates the computationally expensive task of lifting. This is shown in the next theorem.

Theorem 2.3.5. *Let S be an odd hole, odd antihole, web, antiweb or clique existing as an induced sub-hypergraph in a minimal conflict hypergraph H generated from an ISP instance. Let the step size k of the web and antiweb be such that k and d are relatively prime. Then inequality (1) is facet-defining for P^{ISP} if the following condition is satisfied: for any edge $e \in E(H)$ such that $e \setminus V(S) \neq \emptyset$, $e \cap V(S) \cup \{u\}$ is independent in H for every $u \in e \setminus V(S)$*

Proof. From Theorem 2.3.4 we know that inequality (1) is facet-defining for P_S^{ISP}. Therefore, there exist $|V(S)|$ affinely independent vectors that satisfy inequality (1) at equality. Let $I(S) \subset V(S)$ be a maximum vertex packing on S. Now, consider a vertex $v \in V(H) \setminus V(S)$. Let $e \in E(H)$ be an edge such that $v \in e$. Now, if $e \cap V(S) = \emptyset$, then $\{v\} \cup I(S)$ represents a feasible packing whose characteristic vector will satisfy inequality (1) at equality. On the other hand, if $e \cap V(S) \neq \emptyset$, then the condition of the theorem would ensure that $\{v\} \cup I(S)$ still represents a feasible packing in H. Since $v \in V(H) \setminus V(S)$, the characteristic vector of the packing $\{v\} \cup I(S)$ satisfies the inequality (1) at equality.

Therefore, we can get $|V(H) \setminus V(S)|$ such characteristic vectors. It can be easily verified that these vectors, along with the original $|V(S)|$ vectors, are affinely independent. This proves the theorem. $\square$

2.4 Special Cases

We will now state valid inequalities for the specific cases of odd hole, odd antihole, web, antiweb and clique.

Corollary 2.4.1. *Let H be a conflict hypergraph generated from an ISP instance. Let S be an induced sub-hypergraph in H. Let G be the reduced graph associated with S.*

1. *If S is an odd hole, then the following inequality is valid for P^{ISP}*
$$\sum_{j \in V(S)} x_j \leq |V(S)| - \frac{|V(G)|+1}{2} \tag{2}$$
2. *If S is an odd antihole, then the following inequality is valid for P^{ISP}*
$$\sum_{j \in V(S)} x_j \leq |V(S)| - |V(G)| + 2 \tag{3}$$
3. *If S is a web with step size k, then the following inequality is valid for P^{ISP}*
$$\sum_{j \in V(S)} x_j \leq |V(S)| - |V(G)| + k \tag{4}$$
4. *If S is an antiweb with step size k, then the following inequality is valid for P^{ISP}*
$$\sum_{j \in V(S)} x_j \leq |V(S)| - |V(G)| + \left\lfloor \frac{|V(G)|}{k} \right\rfloor \tag{5}$$
5. *If S is a clique, then the following inequality is valid for P^{ISP}*
$$\sum_{j \in V(S)} x_j \leq |V(S)| - |V(G)| + 1 \tag{6}$$

Proof. Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. The following results can be found in previous literature:

1. If G is an odd hole, then the cardinality of the maximum vertex packing on G is $\frac{|V(G)|-1}{2}$ (Padberg [36]).

2. If G is an odd antihole, then the cardinality of the maximum vertex packing on G is 2 (Padberg [36]).

3. If G is a web with step size k, then the cardinality of the maximum vertex packing on G is k (Trotter [41]).

4. If G is an antiweb with step size k, then the cardinality of the maximum vertex packing on G is $\left\lfloor \frac{|V(G)|}{k} \right\rfloor$ (Trotter [41]).

5. If G is a clique, then the cardinality of the maximum vertex packing on G is 1 (Padberg [36]).

The proof now follows from the above results and Theorem 2.3.3. □

Corollary 2.4.2. *Let H be a minimal conflict hypergraph generated from an ISP instance. Let S be an induced sub-hypergraph in H. Let the step size k of the web and antiweb be such that k and d are relatively prime. Then the inequalities given in Corollary 2.4.1 are facet-defining for P_S^{ISP}.*

Proof. The proof follows from Theorem 2.3.4. □

Corollary 2.4.3. *Let H be a minimal conflict hypergraph generated from an ISP instance. Let S be an induced sub-hypergraph in H. Let the step size k of the web and antiweb be such that k and d are relatively prime. Inequalities given in Corollary 2.4.1 will be facet-defining for P^{ISP} if the following condition is satisfied: for any edge $e \in E(H)$ such that $e \setminus V(S) \neq \emptyset$, $e \cap V(S) \cup \{u\}$ is independent in H for every $u \in e \setminus V(S)$*

Proof. The proof follows from Theorem 2.3.5. □

Example 2.4.4.

$$
\begin{aligned}
&2x_1 + 3x_7 \leq 4 \\
&x_1 + x_8 \leq 1 \\
&x_1 + 2x_9 + 8x_{10} \leq 10 \\
&4x_2 + x_3 + 8x_8 \leq 12 \\
&x_2 + x_3 + x_9 + x_{10} \leq 3 \\
&2x_2 + 10x_3 + 2x_{11} \leq 13 \\
&x_4 + x_5 + x_6 + x_9 + x_{10} \leq 4 \\
&x_4 + x_5 + x_6 + x_{11} \leq 3 \\
&x_4 + x_5 + x_6 + x_{12} \leq 3 \\
&2x_7 + x_{11} \leq 2 \\
&4x_7 + 8x_{12} \leq 11 \\
&4x_8 + 5x_{12} \leq 8 \\
&x_j \in \{0, 1\} \quad \forall j = 1, \ldots, 12
\end{aligned}
$$

Consider the ISP instance in Example 2.4.4. Here, variables $\{x_1, x_7\}$ form a dependent set in the first inequality, and therefore constitute an edge in the conflict hypergraph. Similarly, the ninth inequality contains a dependent set formed by the variables $\{x_4, x_5, x_6, x_{12}\}$. Therefore, these variables constitute an edge in the conflict hypergraph. In fact, these inequalities generate a conflict hypergraph that is the same as the one depicted in Figure 6. Thus, we have a web having twelve vertices, eight partitions, and step size three. This gives us the valid inequality $\sum_{j=1}^{12} \leq 7$ (See Corollary 2.4.1). Observe that all the edges of this web are minimal dependent. Hence, this valid inequality is also facet-defining (Corollary 2.4.2) for the web structure. In fact, $(0.25, 0.125, 1, 0, 1, 1, 0.5, 0.75, 0.875, 1, 1, 1)$ is an extreme point in the LP-relaxation polytope that is cut-off by this valid inequality.

Example 2.4.5.

$$4x_1 + 2x_2 + 5x_3 \leq 9$$

$$2x_1 + 8x_4 + x_5 + x_6 \leq 11$$

$$x_2 + x_3 + x_4 + x_5 + x_6 \leq 4$$

$$x_2 + 2x_3 + 3x_7 \leq 5$$

$$3x_4 + 10x_5 + x_6 + x_7 \leq 14$$

$$x_4 + x_5 + x_6 + x_8 \leq 3$$

$$10x_7 + 21x_8 \leq 27$$

$$7x_7 + 9x_9 + 10x_{10} \leq 25$$

$$x_8 + x_9 + x_{10} \leq 2$$

$$8x_8 + x_{11} \leq 8$$

$$x_9 + 2x_{10} + 3x_{11} \leq 5$$

$$x_9 + x_{10} + x_{12} \leq 2$$

$$5x_{11} + 7x_{12} \leq 7$$

$$9x_{11} + x_1 \leq 9$$

$$4x_{12} + 5x_1 \leq 6$$

$$x_{12} + x_2 + x_3 \leq 2$$

$$x_j \in \{0, 1\} \quad \forall j = 1, \ldots, 12$$

Consider the ISP instance shown in Example 2.4.5. The conflict hypergraph generated from inequalities in Example 2.4.5 is the same as the one depicted in Figure 7. Thus, this hypergraph is an antiweb with twelve vertices, eight partitions and step size three. Therefore, we can get the valid inequality $\sum_{j=1}^{12} x_j \leq 6$ (Corollary 2.4.1). Since all the edges are minimal dependent, this valid inequality is facet-defining (Corollary 2.4.2). The extreme point, $(0.913, 1, 0.5, 0.5, 1, 1, 1, 0.5, 0.696, 0.804, 0.898)$, of the LP-relaxation polytope is cut-off by this valid inequality.

In both Examples 2.4.4 and 2.4.5, the conditions of Corollary 2.4.3 are satisfied. Therefore the valid inequalities are facet-defining for the entire polytope (this is also trivially true since the structures themselves constitute the entire conflict hypergraph). However, there are cases when the structure, say, an odd hole, forms only a part of the conflict hypergraph. This is illustrated in Example 2.4.6.

Example 2.4.6.

$$
\begin{aligned}
&7x_1 + 3x_2 + 2x_3 + x_4 + 2x_5 \leq 14\\
&x_4 + x_5 + x_6 \leq 2\\
&x_6 + x_7 + x_8 + x_9 \leq 3\\
&4x_7 + 5x_8 + x_9 + 2x_{10} + 2x_{11} + 3x_{12} \leq 16\\
&x_{10} + x_{11} + x_{12} + x_1 + x_2 + x_3 \leq 5\\
&2x_1 + 4x_2 + 2x_3 + 2x_{13} + 7x_{14} + 3x_{15} \leq 19\\
&x_4 + 2x_5 + 3x_{13} + 4x_{14} + 5x_{15} \leq 14\\
&x_6 + x_{13} + x_{14} + x_{15} \leq 3\\
&15x_7 + 14x_8 + 13x_9 + 9x_{13} + 8x_{14} + 7x_{15} \leq 65\\
&5x_{10} + 7x_{11} + x_{12} + 9x_{13} + 7x_{14} + 7x_{15} \leq 35\\
&x_j \in \{0, 1\} \quad \forall j = 1, \ldots, 15
\end{aligned}
$$

Consider the *ISP* instance shown in Example 2.4.6. The conflict hypergraph induced by inequalities in Example 2.4.6 is shown in Figure 9. Observe that the vertices $\{1, \ldots, 12\}$ form an odd hole. This gives us the valid inequality $\sum_{j=1}^{12} x_j \leq 9$. Since all the edges of the odd hole are minimal dependent, the valid inequality is facet-defining for the odd hole sub-structure. Moreover, the conditions of Corollary 2.4.3 are satisfied. Hence the same inequality is facet-defining for the entire polytope. Further, the valid inequality $\sum_{j=1}^{12} x_j \leq 9$ cuts off an extreme point to the LP relaxation, $(0.33, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0.67, 1, 1, 0.80)$.

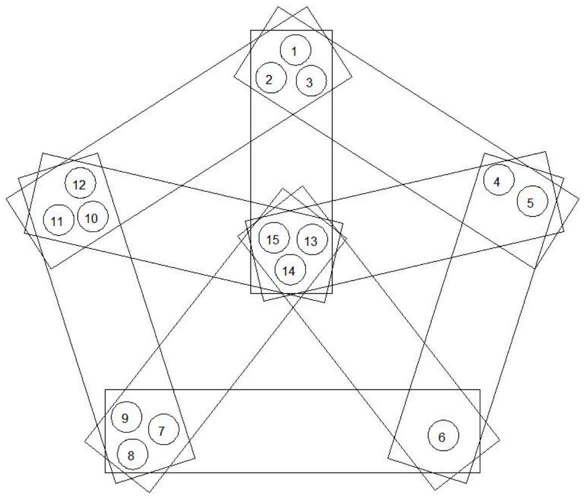


Figure 9: Conflict hypergraph for the ISP instance given in Example 2.4.6.

2.5 Hypergraph Structures With Vertices Outside Edge Intersections

In the definitions of the various hypergraph structures described in Section 2.2, every vertex was in the intersection of two or more edges. In this section, we relax this condition, and explore hypergraph structures that have vertices lying outside of edge intersections.

If all the vertices of a hypergraph structure lie inside edge intersections, then such a structure will be referred to as a *separable* structure. If one or more vertices of a hypergraph structure lie outside edge intersections, then such a structure will be referred to as a *non-separable* structure.

Interestingly, the valid inequalities for non-separable structures are the same as those derived previously for separable structures (Section 2.2). However, we are able to derive facet-defining conditions for odd holes and cliques only.

We will first expand some of the previous definitions.

Definition. 2.5.1 For integer $d \geq 3$, let P^j, $P^{i,j}$, $i, j = 1, \ldots, d$, $i \neq j$ be finite, mutually disjoint, non-empty sets. Let k be an integer such that $1 \leq k \leq \frac{d}{2}$.

1. For odd d, let S be a hypergraph with edge set $E(S) = \{P^j \cup P^{j,j+1} \cup P^{j+1} : j = 1, \ldots, d\}$, where $P^{d+1} \equiv P^1$. Then S is called a *non-separable odd hole.*

2. For odd d, let S be a hypergraph with edge set $E(S) = \{P^i \cup P^{i,j} \cup P^j : |i-j| \geq 2, i = 1, \ldots, d, j = 1, \ldots, d\}$. Then S is called a *non-separable odd antihole.*

3. Let S be a hypergraph with edge set $E(S) = \{P^i \cup P^{i,j} \cup P^j : i, j = 1, \ldots, d, i \neq j\}$. Then S is called a *non-separable clique.*

4. Let S be a hypergraph that has the edge set $E(S) = \{P^i \cup P^{i,j} \cup P^j : j = i+k, \ldots, i+d-k\}$, where $i + l \equiv l$ for all $l = 1, \ldots, d$. Then S is called a *non-separable web.*

5. Let S be a hypergraph that has the edge set $E(S) = \{P^i \cup P^{i,j} \cup P^j : j = 1, \ldots, i+k-1, i+d-k+1, \ldots, d\}$, where $i + l \equiv l$ for all $l = 1, \ldots, d$. Then S is called a *non-separable antiweb.*

We will refer to P^j's as *end partitions* and $P^{i,j}$'s as *middle partitions* of S, $i, j = 1, \ldots, d$. For the last two definitions, we will call k the *step size* of the non-separable web or the antiweb.

Observe that in the case when all the middle partitions are empty, we get the separable structures that were described earlier.

As described in the previous sections, a reduced graph may be constructed for each of these structures. Let G be a graph with vertex set $V(G) = \{1, \ldots, d\}$ and edge set $E(G) = \{\{i, j\} : P^i \cup P^{i,j} \cup P^j \in E(S)\}$. Then G is the reduced graph associated with S.

Lemma 2.5.2. *Let S be a non-separable odd hole, odd antihole, web, antiweb or clique. Let G be the associated reduced graph. Let $I(G) \subset V(G)$ be a maximum vertex packing on G. Let $I(S) \subset V(S)$ be a packing on S such that all the middle partitions of S contribute all their vertices. Then $|I(S)| \leq |V(S)| - |V(G)| + |I(G)|$.*

Proof. We begin the proof by showing that, in a maximum vertex packing, any partition (end or middle) must contribute either all or all-but-one of its vertices. To show this, suppose there exists a packing that is maximum and where one of the partitions contributes all-but-two of its vertices. In such a case, we can always select one more vertex from that partition because of the mutual disjointness of partitions, thereby allowing us to increase the cardinality of this packing. But this contradicts the assumption that the packing was maximum. This shows that all the partitions of a maximum vertex packing contribute all or all-but-one of their vertices. Now, consider the vertex packing $I(S) = \{\bigcup_{j\in I(G)} P^j\} \bigcup \{\bigcup_{\{ij\}\in E(G)} P^{ij}\} \bigcup \{\bigcup_{j\in V(G)\setminus I(G)} \bar{P}^j\}$, where $\bar{P}$ represents the set obtained by removing one element of the set P.

To prove the lemma, consider an arbitrary vertex packing $I'(S)$ where all the middle partitions contribute all of their vertices. Suppose $|I'(S)| > |I(S)|$. We showed that in a maximum vertex packing all the partitions contribute either all or all-but-one of their vertices. This fact, along with the supposition $|I'(S)| > |I(S)|$ implies that a greater number of end partitions must be contributing all of their vertices in $I'(S)$ (since all the middle partitions in $I(S)$ as well as $I'(S)$ contribute all of their vertices).

Now, because of the correspondence between the vertices of G and the end partition of S, the set $I'(G) = \{j : P_j$ contributes all of its vertices in $I(S)\}$ must represent a feasible packing on G. Since a greater number of end partitions contribute all of their vertices in $I(S)$, we can infer that $|I'(G)| > |I(G)|$. But this contradicts the assumption that $I(G)$ is a maximum vertex packing on G. This contradiction proves the lemma that the cardinality of any vertex packing on S where the all the middle contribute all of their vertices can never exceed $|I(S)|$.

It can be easily verified that $|I(S)| = |V(S)| - |V(G)| + |I(G)|$. □

Proposition 2.5.3. *Let S be a non-separable odd hole, odd antihole, web, antiweb or clique. Let G be the reduced graph derived from S. Let $I(G) \subset V(G)$ be a maximum* *vertex packing on G. Then the cardinality of maximum vertex packing on S is $|V(S)| - |V(G)| + |I(G)|$*

Proof. Let $I(S) = \{\bigcup_{j \in I(G)} P^j\} \bigcup \{\bigcup_{\{ij\} \in E(G)} P^{ij}\} \bigcup \{\bigcup_{j \in V(G) \setminus I(G)} \bar{P}^j\}$, where $\bar{P}$ represents the set obtained by removing one element of the set P. We claim that $I(S)$ represents a maximum vertex packing on S. It can be easily verified that $|I(S)| = |V(S)| - |V(G)| + |I(G)|$.

To prove the claim, consider an arbitrary vertex packing where all the middle partitions contribute all their vertices. Lemma 2.5.2 implies that the cardinality of a vertex packing where all the middle partitions contribute all their vertices can never exceed $|I(S)|$.

Now consider a packing $I'(S)$ where some of the middle partitions contribute all-but-one of their vertices. Let P^{ij} be a middle partition that contributes all-but-one of its vertices in $I'(S)$. The mutual disjointness of all the partitions and the definitions of the hypergraph structures imply that a middle partition is connected to exactly two end partitions. Among the two end partitions (P^i and P^j) connected to P^{ij}, at least one must contribute all of it vertices. If this were not true, then we could increase the cardinality of $I'(S)$ simply by selecting one more vertex from P^{ij}. Let P^i be the partition that contributes all of its vertices. Now, we will modify $I'(S)$ such that P^{ij} contributes all of its vertices and P^i contributes all-but-one of its vertices. Clearly, this does not make the packing infeasible, and the cardinality remains the same. Because, we picked the middle partition arbitrarily, we can perform this operation for all the middle partitions that contribute all-but-one of their vertices. Let $I''(S)$ be the new packing that emerges after this operation. Clearly, $|I''(S)| = |I'(S)|$. Therefore, $|I''(S)| > |I(S)|$ (by assumption). But, all the middle partitions in $I''(S)$ contribute all of their vertices. Thus the maximality of $I(S)$ is violated. This contradiction proves that $I(S)$ is indeed a maximum vertex packing on S.

This completes the proof. □

Proposition 2.5.4. *Let S be a non-separable odd hole or clique. Then there exist $|V(S)|$ maximum vertex packings on S with affinely independent characteristic vectors.*

Proof. Let $P^1, \ldots, P^d$ denote the end partitions of S. Let G be the reduced graph associated with S. Let x^1 be a characteristic vector of a maximum vertex packing on G such that $x^1_1 = 0$. Construct x^i such that $x^{i+1}_{j+1} = x^i_j$, $i, j = 1, \ldots, d$, where index $d + 1 \equiv 1$. Therefore, $x^1, \ldots, x^d$ are affinely independent characteristic vectors representing maximum vertex packings on G, and $x^i_i = 0$ for all $i = 1, \ldots, d$. Let $X = \{x^1, \ldots, x^d\}$. For any (end or middle) partition P, let the elements of P be indexed such that P_j denotes the j^{th} element of P, $j = 1, \ldots, |P|$. Now, for a given $i \in \{1, \ldots, d\}$, let $Y^i \in \{0, 1\}^{|V(H)|}$ such that

$$Y^i = \{\sum_{x^i_j=1} u_{P^j} + \sum_{x^i_j=0, j\neq i} u_{P^j\setminus\{P^j_1\}} + u_{P^i\setminus\{P^i_l\}} + \sum_{\{i,j\}\in E(G)} u_{P^{i,j}} : l = 1, \ldots, |P^i|\}$$

For every vertex that belongs to an end partition of S, the collection $Y = \{Y^i, \ldots, Y^d\}$ contains a characteristic vector that that represents a maximum vertex packing on S. Moreover, all these points are affinely independent.

To obtain the rest of the affinely independent points, we will need the following. Let $P^{i,j}$ be a middle partition of S, with P^i, P^j the associated end partitions. That is, $\{P^i \cup P^{i,j} \cup P^j\} \in E(S)$. For every such middle partition $P^{i,j}$, we need a maximum vertex packing on S such that either P^i or P^j satisfies the following condition:

> The end partition P^i (or P^j) contributes all-but-one of its vertices, and every partition P^k such that P^k, P^i (or P^k, P^j) belong to the same edge, $k \neq j$ (or $k \neq i$), contributes all-but-one of its vertices.

Observe that these conditions are satisfied for all middle partitions for cliques and odd holes, as shown below.

Consider an odd hole with an edge $\{P^i \cup P^{i,i+1} \cup P^{i+1}\}$, $i \in \{1, \ldots, d\}$. Because of the structure of odd holes, we can construct a maximum vertex packing such that the partitions $P^{i+3}, P^{i+5}, \ldots, P^{i+d-1}$ contribute all their vertices, and the rest of the partitions contribute all-but-one of their vertices. Therefore, P^{i+d-1} contributes all its vertices, and the partitions P^i and P^{i+1} contribute all-but-one of their vertices. Apart from P^{i+d-1} (which contributes all its vertices), the only partition P^i is connected to is P^{i+1}. Both P^i and P^{i+1} contribute all-but-one of their vertices. Therefore, the conditions are satisfied. Because the edge was chosen arbitrarily, all the middle partitions of the odd hole will satisfy these conditions.

Now consider a clique with an edge $\{P^i \cup P^{i,j} \cup P^j\}$, where $i, j \in \{1, \ldots, d\}$, $i \neq j$. Because of the special structure of cliques, we can construct a maximum vertex packing such that P^i contributes all of its vertices, and all the other partitions contribute all-but-one of their vertices. This means P^j contributes all-but-one of its vertices, and all the partitions (excluding P^i) connected to it contribute all-but-one of their vertices. Therefore, the condition prescribed above is satisfied for cliques too.

Now, for every $\{i, j\} \in E(G)$, we define the collection of packings $X^{i,j} = \{x : x \in X, x_i = 1, x_j = 0, x_k = 0 \ \forall \ \{j, k\} \in E(G), k \neq i\}$. Now, for $\{i, j\} \in E(G)$, let $Z^{i,j} \in \{0, 1\}^{|V(S)|}$ such that

$$Z^{i,j} = \{\sum_{x_k=1} u_{P^k} + \sum_{x_k=0, k\neq j} u_{P^k \setminus \{P^k_1\}} + u_{P^j} + \sum_{\{m,n\} \in E(G) \setminus \{i,j\}} u_{P^{m,n}} + u_{P^{i,j} \setminus P^{i,j}_l} : \\ l = 1, \ldots, |P^{i,j}|\}$$

where the packing $x \in X^{i,j}$.

For every vertex belonging to a middle partition, the collection $Z = \{Z^{i,j} : \{i, j\} \in E(G)\}$ contains a characteristic vector that represents a maximum vertex packing on S. Moreover, all these points are affinely independent.

The collections Y and Z together contain $|V(S)|$ feasible, affinely independent points that represent maximum vertex packings on S. This completes the proof. $\square$

Theorem 2.5.5. *Let H be a conflict hypergraph generated from an ISP instance. Let S be a non-separable odd hole, odd antihole, web, antiweb or clique existing as an induced sub-hypergraph in H. Let G be the reduced graph derived from S. Let $I(G) \subset V(G)$ be a maximum vertex packing on G. Then the following is a valid inequality for P^{ISP}:*

$$\sum_{j \in V(S)} x_j \leq |V(S)| - |V(G)| + |I(G)| \tag{7}$$

Proof. Proof follows from Proposition 2.5.3. $\square$

Theorem 2.5.6. *Let H be a minimal conflict hypergraph generated from an ISP instance. Let S be a non-separable odd hole or clique existing as an induced sub-hypergraph in H. Then the inequality (7) is facet-defining for P_S^{ISP}.*

Proof. Consider the unit vectors u_j, $j \in V(S)$, along with the 0 vector. These $|V(S)|+1$ vectors are affinely independent and satisfy the constraints of P_S^{ISP}. Therefore, $dim(P_S^{ISP}) = |V(S)|$. Thus, to prove that the inequality is facet-defining, we need to show that there exist $|V(S)|$ feasible, affinely independent points that satisfy inequality (7) at equality.

The vectors constructed in Proposition 2.5.4 satisfy these conditions for non-separable clique and odd hole. $\square$

Observe that the valid inequalities remain the same for both the non-separable and the separable cases. However, facet-defining conditions are more restrictive for non-separable structures. Theorem 2.5.6 shows that non-separable odd holes and cliques can induce facets for P_S^{ISP}. The following theorem shows that the non-separable odd hole and clique can also induce facets for P^{ISP} under certain conditions.

Theorem 2.5.7. *Let S be a non-separable odd hole or clique existing as an induced sub-hypergraph in a minimal conflict hypergraph H generated from an ISP instance.*

Inequality (7) is facet-defining for P^{ISP} if the following condition is satisfied: for any edge $e \in E(H)$ such that $e \setminus V(S) \neq \emptyset$, $e \cap V(S) \cup \{u\}$ is independent in H for every $u \in e \setminus V(S)$

Proof. From Theorem 2.5.6 we know that inequality (7) is facet-defining for P_S^{ISP} if S is a clique or an odd hole. Thus, there exist $|V(S)|$ affinely independent vectors that satisfy inequality (7) at equality. Let $I(S) \subset V(S)$ be a maximum vertex packing on S. Now, consider a vertex $v \in V(H) \setminus V(S)$. Let $e \in E(H)$ be an edge such that $v \in e$. Now, if $e \cap V(S) = \emptyset$, then $\{v\} \cup I(S)$ represents a feasible packing (in H) whose characteristic vector will satisfy inequality (7) at equality. On the other hand, if $e \cap V(S) \neq \emptyset$, then the condition specified in the theorem would ensure that $\{v\} \cup I(S)$ still represents a feasible packing in H. Since $v \in V(H) \setminus V(S)$, the characteristic vector of the packing $\{v\} \cup I(S)$ satisfies the inequality (7) at equality.

Therefore, we can get $|V(H) \setminus V(S)|$ such characteristic vectors. It can be easily verified that these vectors, along with the original $|V(S)|$ vectors, are affinely independent. This proves the theorem. □

Corollary 2.5.8. *Let H be a conflict hypergraph generated from an ISP instance. Let S be an induced sub-hypergraph in H. Let G be the reduced graph associated with S.*

1. *If S is a non-separable odd hole, then the following inequality is valid for P^{ISP}*

$$\sum_{j \in V(S)} x_j \leq |V(S)| - \frac{|V(G)| + 1}{2} \tag{8}$$

2. *If S is a non-separable odd antihole, then the following inequality is valid for P^{ISP}*

$$\sum_{j \in V(S)} x_j \leq |V(S)| - |V(G)| + 2 \tag{9}$$

3. *If S is a non-separable web with step size k, then the following inequality is valid for P^{ISP}*

$$\sum_{j \in V(S)} x_j \leq |V(S)| - |V(G)| + k \tag{10}$$

4. *If S is a non-separable antiweb with step size k, then the following inequality is valid for P^{ISP}*

$$\sum_{j \in V(S)} x_j \leq |V(S)| - |V(G)| + \left\lfloor \frac{|V(G)|}{k} \right\rfloor \tag{11}$$

5. *If S is a non-separable clique, then the following inequality is valid for P^{ISP}*

$$\sum_{j \in V(S)} x_j \leq |V(S)| - |V(G)| + 1 \tag{12}$$

Proof. Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. The following results can be found in previous literature:

1. If G is a separable odd hole, then the cardinality of the maximum vertex packing on G is $\frac{|V(G)|-1}{2}$ (Padberg [36]).

2. If G is a separable odd antihole, then the cardinality of the maximum vertex packing on G is 2 (Padberg [36]).

3. If G is a separable web with step size k, then the cardinality of the maximum vertex packing on G is k (Trotter [41]).

4. If G is a separable antiweb with step size k, then the cardinality of the maximum vertex packing on G is $\left\lfloor \frac{|V(G)|}{k} \right\rfloor$ (Trotter [41]).

5. If G is a separable clique, then the cardinality of the maximum vertex packing on G is 1 (Padberg [36]).

The proof now follows from the above results and Theorem 2.5.5. □

Corollary 2.5.9. *Let H be a minimal conflict hypergraph generated from an ISP instance. Let S be an induced sub-hypergraph in H. Then inequalities (8) and (12) are facet-defining for P_S^{ISP}.*

Proof. The proof follows from Theorem 2.5.6. $\square$

Corollary 2.5.10. *Let H be a minimal conflict hypergraph generated from an ISP instance. Let S be an induced sub-hypergraph in H. Then inequalities (8) and (12) will be facet-defining for P^{ISP} if the following condition is satisfied: for any edge $e \in E(H)$ such that $e \setminus V(S) \neq \emptyset$, $e \cap V(S) \cup \{u\}$ is independent in H for every $u \in e \setminus V(S)$*

Proof. The proof follows from Theorem 2.5.7. $\square$

Example 2.5.11.

$$
\begin{aligned}
&2x_1 + 4x_2 + 3x_3 + x_4 + x_5 \leq 10\\
&x_1 + x_{19} + x_{20} \leq 2\\
&x_4 + x_5 + x_6 + x_7 \leq 3\\
&3x_{19} + 2x_{18} + 5x_{11} \leq 9\\
&2x_6 + 3x_7 + x_{11} \leq 5\\
&x_{10} + x_{11} \leq 1\\
&x_{10} + 2x_{12} + 3x_{17} + 4x_{19} \leq 9\\
&x_{10} + x_{13} \leq 1\\
&x_{13} + x_{16} \leq 1\\
&x_{13} + x_{14} + x_{15} \leq 2\\
&2x_6 + 5x_7 + 10x_8 + 7x_9 \leq 23\\
&x_9 + x_{11} \leq 1\\
&23x_6 + 13x_7 + 17x_{10} \leq 50\\
&29x_9 + 84x_{10} \leq 86\\
&x_j \in \{0,1\} \quad \forall j = 1, \ldots, 20
\end{aligned}
$$

Consider the ISP instance shown in Example 2.5.11. The conflict hypergraph generated by the inequalities in Example 2.5.11 is shown in Figure 10. The vertices in the set $V_1 = \{1,2,3,4,5,6,7,11,18,19,20\}$ comprise a non-separable odd hole with five edges. Therefore, the inequality $\sum_{j \in V_1} x_j \leq 8$ is valid. All the edges of this odd hole are minimal dependent. Therefore, the inequality is facet-defining for the odd hole induced sub-hypergraph. Similarly, there is a non-separable clique comprised of the vertices in the set $V_2 = \{6,7,8,9,10,11\}$. Therefore, inequality $\sum_{j \in V_2} x_j \leq 3$ is valid. It is also facet-defining for the entire convex hull.

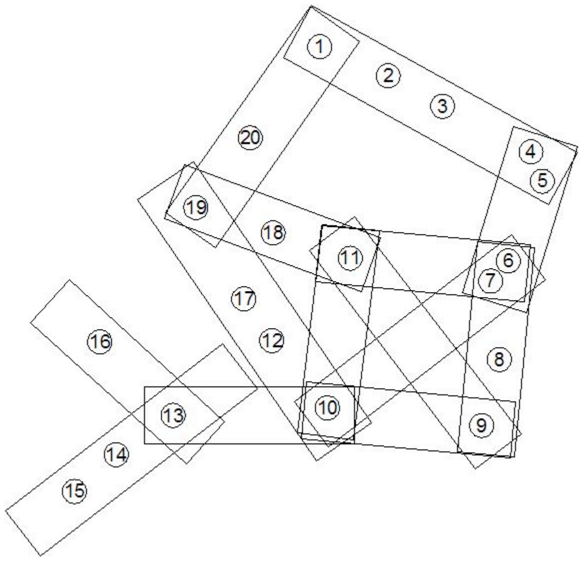


Figure 10: Conflict hypergraph for the *ISP* instance given in Example 2.5.11.

2.6 Chvatal-Gomory Ranks of Hypergraph Inequalities

In this section, we investigate the *Chvatal-Gomory (C-G)* ranks of the valid inequalities we derived earlier. An inequality $\pi x \leq \pi_0$, valid for P^{ISP}, is said to be of rank k if it is not equivalent to or dominated by any non-negative linear combination of valid inequalities, each of which can be determined by no more than $k-1$ applications of the C-G procedure. (See Nemhauser and Wolsey [35] for a detailed exposition on C-G ranks.)

Let S be a non-separable odd hole or clique. Let G be the associated reduced graph. Let $I(G) \subset V(G)$ be a maximum vertex packing on G.

Theorem 2.6.1. *The rank of the odd hole inequality* $\sum_{j \in V(S)} x_j \leq |V(S)| - \frac{|V(G))|+1}{2}$ *is 1.*

Proof. The odd hole inequality in a conflict hypergraph can be written as

$$\sum_{j=1}^{d}(\sum_{i \in P^j} x_i) + \sum_{j}^{d}(\sum_{i \in P^{j,j+1}} x_i) \leq \sum_{j=1}^{d}(|P^j| - 1) + \sum_{j=1}^{d}|P^{j,j+1}| + \frac{d-1}{2} \qquad (13)$$

where $d = |V(G)|$.

We will show that this inequality can be obtained by applying a single round of the C-G procedure on rank 0 inequalities. Consider the following rank 0 inequalities associated with the edges in a conflict hypergraph:

$$\sum_{i\in P^j} x_i + \sum_{i\in P^{j,j+1}} x_i + \sum_{i\in P^{j+1}} x_j \le (|P^j|-1) + |P^{j,j+1}| + (|P^{j+1}|-1) + 1 \quad \forall j = 1,\dots,d$$
$$\sum_{i\in P^{j,j+1}} x_i \le |P^{j,j+1}| \quad \forall j = 1,\dots,d \tag{14}$$

Adding them together, we get the inequality

$$2\sum_{j=1}^{d}(\sum_{i\in P^j} x_i) + 2\sum_{j}^{d}(\sum_{i\in P^{j,j+1}} x_i) \le 2\sum_{j=1}^{d}(|P^j|-1) + 2\sum_{j=1}^{d}|P^{j,j+1}| + d \tag{15}$$

Dividing both sides by 2 and rounding down, as per the C-G procedure, we obtain the odd hole inequality. We obtained the odd hole inequality by applying exactly one round of the C-G procedure on rank 0 inequalities. Therefore, the odd hole inequality has rank 1.

□

Theorem 2.6.2. *The rank of the clique inequality* $\sum_{j\in V(S)} x_j \le |V(S)| - |V(G))| + |I(G)|$ *is* $O(log(|V(G)|)$.

Proof. The clique inequality in a conflict hypergraph can be written as

$$\sum_{j=1}^{d}\left(\sum_{i\in P^j} x_i\right) + \sum_{j,k=1,\dots,d,j\ne k}\left(\sum_{i\in P^{j,k}} x_i\right) \le$$
$$\sum_{j=1}^{d}\left(|P^j|-1\right) + \sum_{j,k=1,\dots,d,j\ne k}|P^{j,k}| + 1 \tag{16}$$

where $|V(G)| = d$.

We claim that the rank of inequality (16) is ρ if $2^{\rho-1}+2 \le d \le 2^{\rho}+1$, $\rho \in \mathbb{Z}_+$. This is proven by induction on ρ. When $\rho = 1$, $2^{\rho-1}+2 = 3 \le 2^{\rho}+1 = 3$. Therefore, $d = 3$. Hence, according to the claim, the rank of inequality (16) is 1 for $d = 3$. This can be shown by considering the following edge inequalities:

$$\begin{aligned}
\sum_{i\in P^1} x_i + \sum_{i\in P^{1,2}} x_i + \sum_{i\in P^2} x_i &\le (|P^1|-1) + |P^{1,2}| + (|P^2|-1) + 1\\
\sum_{i\in P^{1,2}} x_i &\le |P^{1,2}|\\
\sum_{i\in P^2} x_i + \sum_{i\in P^{2,3}} x_i + \sum_{i\in P^3} x_i &\le (|P^2|-1) + |P^{2,3}| + (|P^3|-1) + 1\\
\sum_{i\in P^{2,3}} x_i &\le |P^{2,3}|\\
\sum_{i\in P^3} x_i + \sum_{i\in P^{3,1}} x_i + \sum_{i\in P^1} x_i &\le (|P^3|-1) + |P^{3,1}| + (|P^1|-1) + 1\\
\sum_{i\in P^{3,1}} x_i &\le |P^{3,1}| \qquad (17)
\end{aligned}$$

Adding the inequalities (17), we get

$$\begin{aligned}
2\sum_{j=1}^{3}\left(\sum_{i\in P^j} x_i\right) + 2\sum_{j,k=1,2,3,j\neq k}\left(\sum_{i\in P^{j,k}} x_i\right) \le\\
2\sum_{j=1}^{3}\big(|P^j|-1\big) + 2\sum_{j,k=1,2,3,j\neq k}|P^{j,k}| + 3 \qquad (18)
\end{aligned}$$

Dividing inequality (18) by 2, and then rounding down according to the C-G procedure, we get the clique inequality. We applied the C-G procedure exactly once on the rank 0 inequalities. This establishes that the rank is 1 for $\rho = 1$.

Suppose the claim holds for $\rho \le u$; that is, inequality (16) is of rank ρ if $2^{\rho-1} + 2 \le d \le 2^{\rho} + 1$ for all $\rho \le u$. Now, consider the case $\rho = u + 1$. Therefore, $2^{\rho-1} + 2 = 2^u + 2 \le d \le 2^{u+1} + 1 = 2^{\rho} + 1$.

Now, observe that any two end partition in a clique lie in the same edge. Therefore, any subset of end partitions of a clique also form a clique. This implies that out the d end partitions of a clique, any combination of $2^u + 1$ end partitions also form a clique. This property of cliques gives us the following set of valid inequalities:

$$\sum_{j\in Q}\left(\sum_{i\in P^j} x_i\right) + \sum_{j,k\in Q, j\neq k}\left(\sum_{i\in P^{j,k}} x_i\right) \leq \sum_{j\in Q}(|P^j|-1) + \sum_{j,k\in Q, j\neq k}|P^{j,k}| + 1$$
$$\forall Q \subset \{1,\ldots,d\}, |Q| = 2^u+1 \qquad (19)$$

From the induction hypothesis, each of the inequalities (19) have rank u. Note that there are $\binom{d}{2^u+1}$ such inequalities. In the inequalities (19), each end partition term occurs $\binom{d-1}{2^u}$ times, and each middle partition term occurs $\binom{d-2}{2^u-1}$ times.

Now, $\frac{\binom{d-2}{2^u-1}}{\binom{d-1}{2^u}} = \frac{2^u}{d-1} < 1$. Therefore, $\binom{d-1}{2^u} > \binom{d-2}{2^u-1}$. Let $\alpha = \binom{d-1}{2^u} - \binom{d-2}{2^u-1}$.

We also have the following valid inequalities (each of rank 0):

$$\alpha \sum_{j,k\in Q, j\neq k}\left(\sum_{i\in P^{j,k}} x_i\right) \leq \alpha \sum_{j,k\in Q, j\neq k}|P^{j,k}|$$
$$\forall Q \subset \{1,\ldots,d\}, |Q| = 2^u+1 \qquad (20)$$

Adding inequalities (19) and (20), we get

$$\binom{d-1}{2^u}\sum_{j=1}^{d}\sum_{i\in P_j} x_i + \binom{d-1}{2^u}\sum_{j,k=1,\ldots,d, j\neq k}\sum_{i\in P^{j,k}} x_i \leq$$
$$\binom{d-1}{2^u}\sum_{j=1}^{d}(|P^j|-1) + \binom{d-1}{2^u}\sum_{j,k=1,\ldots,d, j\neq k}|P^{j,k}| + \binom{d}{2^u+1} \qquad (21)$$

Dividing both sides by $\binom{d-1}{2^u}$ we get

$$\sum_{j=1}^{d}\sum_{i\in P_j} x_i + \sum_{j,k=1,\ldots,d, j\neq k}\sum_{i\in P^{j,k}} x_i \leq$$
$$\sum_{j=1}^{d}(|P^j|-1) + \sum_{j,k=1,\ldots,d, j\neq k}|P^{j,k}| + \frac{d}{2^u+1} \qquad (22)$$

Now, $2^u+2 \leq d \leq 2^{u+1}+1$. Therefore $\frac{2^u+2}{2^u+1} \leq \frac{d}{2^u+1} \leq \frac{2^{u+1}+1}{2^u+1}$, or $1 + \frac{1}{2^u+1} \leq \frac{d}{2^u+1} \leq 1 + \frac{2^u}{2^u+1}$. Therefore, $\left\lfloor \frac{d}{2^u+1} \right\rfloor = 1$.

Therefore, after rounding down, (22) reduces to the form represented by (16). This shows that the rank is at most $u+1$.

To show that the rank is at least $u+1$, we merely need to show that the valid inequality just obtained is not dominated by any inequality of rank u or less.

Let x^* be the characteristic vector representing a feasible solution to the LP-relaxation of a VPP instance for this clique (S) such that

$$x^*_j = \begin{cases} 1 & \text{if } j \in P^i \setminus \{P^i_1\}, i = 1, \ldots, d \\ \frac{1}{2^u+1} & \text{if } j = P^i_1, i = 1, \ldots, d \\ 1 & \text{if } j \in P^{ik}, i, k = 1, \ldots, d, i \neq k \end{cases}$$

x^* satisfies the inequality (16) when $d \leq 2^u + 1$. But it does not satisfy (16) when $2^u + 2 \leq d \leq 2^{u+1} + 1$. Hence the rank is at most $u + 1$. This completes the proof of the claim.

Now, according to the claim, the rank of the clique inequality (16) is ρ if $2^{\rho-1}+2 \leq d \leq 2^\rho + 1$, $\rho \in \mathbb{Z}_+$. Therefore, $\rho - 1 \leq log(d) \leq \rho$. Hence the claim implies that the rank ρ is $O(log(d))$. This proves the theorem.

□

Observe that the Chvatal-Gomory ranks remain unchanged for odd hole and clique inequalities. It is our conjecture, based on the correspondence between elements of the hypergraph structures and the associated reduced graphs, that Chvatal-Gomory ranks for other hypergraph inequalities should be same as those of graph structures.

2.7 Separation

2.7.1 Introduction

We will now investigate separation algorithms for the hypergraph structures described in the preceding sections. We will first discuss the general separation problem in conflict graphs. We will then discuss how to extend separation algorithms from conflict graphs to conflict hypergraphs.

For an ISP instance and an $x^* \notin P^{ISP}$, the solution to a separation problem instance gives a valid inequality gives a valid inequality $\pi x \leq \pi_0$ such that $\pi x^* > \pi_0$. If no such inequality can be obtained, then it should imply that none exist. It can be shown that the complexity of separation is equivalent to optimization. (see Nemhauser and Wolsey [35] for details).

In the context of conflict graphs, a separation algorithm identifies a graph structure (like clique or odd hole) such that the violation, for a given solution to the LP relaxation, of the associated valid inequality is maximized. The algorithm is designed such that the graph structure returned by the algorithm conforms to the definition of the structure. Thus, a separation algorithm for cliques, for example, would return a set of vertices from the conflict graph such that each vertex is connected to every other vertex, and the violation of the associated clique inequality is maximized. The separation algorithm for hypergraphs is similar. The hypergraph structure returned by the separation algorithm should conform to its original definition, and the violation of the associated valid inequality should be maximized.

In the conflict graph, a given structure's definition rests solely upon the connectivity among the vertices. In a conflict hypergraph, the definition of the structures depends on connectivity among the end partitions. In addition, all the partitions must be mutually disjoint. A separation algorithm for conflict hypergraphs must take this into account.

Our basic strategy will be to construct a reduced graph from the conflict hypergraph, and use graph algorithms and heuristics to separate graph structures on the reduced graph. We can then use the relationship between the reduced graph and the conflict hypergraph to obtain the hypergraph structure from this graph structure.

2.7.2 Constructing the Reduced Graph

Let $V(H)$ be the set of vertices of the conflict hypergraph, and let $E(H)$ be the set of edges. We define the collection of end partitions as $\mathbb{P}_E = \{\bigcap_{e \in M} e : M \subset$

$2^{E(H)}, |M| \geq 2\}$. Thus, an end partition is an intersection of two or more edges. Similarly, let $\mathbb{P}_M = \{P^{\{i,j\}_e} = e \setminus \{P^i \cup P^j\} : P^i, P^j \in \mathbb{P}_E, P^i, P^j \subset e, e \in E(H)\}$ be the set of middle partitions.

We will now create a graph G as follows. For every end partition $P^i \in \mathbb{P}_E$, there exist vertices $i \in V(G)$. If any two end partitions $P^i, P^j \in \mathbb{P}_E$ are contained in the same edge $e \in E(H)$, then we create edges $\{i, j\}_e \in E(G)$. The subscript for the edge $e \in E(H)$ is necessary because there might be more than one hypergraph edges that contain both P^i and P^j.

Thus, every vertex in the graph G corresponds to an end partition in the hypergraph H, and every edge in the graph G corresponds to an edge in the hypergraph H.

2.7.3 Valid Structures

Observe that a structure that might be valid on the reduced graph may not be valid in the conflict hypergraph. This is due to the following two requirements that need to be satisfied in order to ensure the validity of the hypergraph structures:

1. All partitions must be mutually disjoint

2. An edge contains exactly two end partitions

Therefore, for reduced graph structure S to represent a valid hypergraph structure, for any $\{i, j\}_{e_1}, \{k, l\}_{e_2} \in E(S)$, the partitions $P^i, P^j, P^{\{i,j\}_{e_1}}, P^k, P^l, P^{\{k,l\}_{e_2}}$ must be mutually disjoint.

A number of researchers have investigated the separation problem of graph structures on conflict graphs. See [4], [10] and [25] for examples. We will now illustrate this concept by extending a separation algorithm for odd cycles in directed graphs (described in [25]) to hypergraphs.

2.7.4 Separation Algorithm for Odd Cycles

In this section, we investigate the separation problem for the case of odd cycles. We begin by describing the graph algorithm given in [25]. We will then show how this algorithm can be extended to the case of conflict hypergraph and odd cycles therein.

2.7.4.1 Detecting Minimum Weight Odd Cycles in Directed Graphs

Here we summarize the algorithm for detecting minimum weight odd cycles in directed graphs. The details of the algorithm can be found in [25]. Consider a directed graph D with node set $N(D)$ and arc set $A(D)$. Let there be a non-negative weight $w_{(i,j)}$ associated with every arc $(i, j) \in A(D)$. To detect a minimum weight odd cycle on this graph, create a new directed graph D' with node set $N(D')$ such that for every $i \in N(D)$, there exists $i, i' \in N(D')$. The set of arcs $A(D')$ for the new directed graph is constructed in the following manner. For every arc $(i, j) \in A(D)$, the new directed graph contains arcs $(i, j'), (i', j) \in A(D')$. The weights $w_{(i,j')}, w_{(i',j)}$ of the arcs $(i, j'), (i', j) \in A(D')$ are same as the weight $w_{(i,j)}$ of the original arc $(i, j) \in A(D)$. Now, in the directed graph D', any path from node i to i' represents an odd cycle on the original directed graph D. Thus, finding the minimum weight odd cycle on (N, A) entails the solution of at most $|N(D)|$ shortest path problems on D'.

2.7.4.2 Separating Odd Cycles on Conflict Hypergraphs

Let H be the conflict hypergraph generated by an ISP instance. Let $V(H)$ be the set of vertices of the conflict hypergraph, and let $E(H)$ be the set of edges. Let x^* be the optimal solution to the current LP relaxation.

The separation strategy works in the following manner. We will first construct the reduced graph in the manner described earlier. Since the reduced graph has undirected edges, we will convert it into a directed graph (by creating two opposite directed arcs for every undirected edge). We will then create another directed graph so that the odd cycle problem can be modeled as a shortest path problem.

Let $\mathbb{P}_E = \{\bigcap_{e \in M} e : M \subset 2^{E(H)}, |M| \geq 2\}$ be the collection of end partitions (intersection of two or more edges). Let $\mathbb{P}_M = \{P^{\{i,j\}_e} = e \setminus \{P^i \cup P^j\} : P^i, P^j \in \mathbb{P}_E, P^i, P^j \subset e, e \in E(H)\}$ be the set of middle partitions.

We construct the reduced graph G as described previously. For every end partition $P^i \in \mathbb{P}_E$, there exists a verticex $i \in V(G)$. If any two end partitions $P^i, P^j \in \mathbb{P}_E$ are contained in the same edge $e \in E(H)$, then we create edges $\{i,j\}_e \in E(G)$.

Thus, every vertex in the graph G corresponds to an end partition in the hypergraph H, and every edge in the graph G corresponds to an edge in the hypergraph H.

The reduced graph G is an undirected graph. We will convert G into a directed graph D. We will refer to the node set of D with $N(D)$ and the arc set with $A(D)$. Let $N(D) = V(G)$. For every $\{i,j\}_e \in E(G)$, create $(i,j)_e, (j,i)_e \in A(D)$.

We now create another directed graph D', with node set $N(D')$ and arc set $A(D')$, in the following manner. For every $i \in N(D)$, create $i, i' \in N(D')$. For every $(i,j)_e \in A(D)$, create $(i,j')_e, (i',j)_e \in A(D')$. Observe that every node in the directed graph D' corresponds to an end partition in the hypergraph H, and every arc corresponds to an edge in the hypergraph H. For any $(j,k)_e \in A(D')$, let $P^{\{j,k\}_e} = e \setminus \{P^j \cup P^k\}$.

We will now formulate a separation problem for the odd cycles. The formulation is essentially a shortest path problem with the added constraints for ensuring mutual disjointness of the associated partitions.

For a given feasible solution to the ISP instance x^*, let

$$w_{(i,j)_e} = \frac{1}{2}\sum_{k \in P^i}(1 - x_k^*) + \sum_{k \in P^{\{i,j\}_e}}(1 - x_k^*) + \frac{1}{2}\sum_{k \in P^j}(1 - x_k^*)$$

where $(i,j)_e \in A(D')$, $P^i, P^j \in \mathbb{P}_E$.

We define the following decision variables. For every $(i,j)_e \in A(D')$, let

$$x_{(i,j)_e} = \begin{cases} 1 & \text{if } (i,j)_e \in A(D') \text{ is selected} \\ 0 & \text{otherwise.} \end{cases}$$

We also define the following constants.

For every $j \in N(D')$, let

$$\alpha_j = \begin{cases} 1 & \text{if } j = i \\ -1 & \text{if } j = i' \\ 0 & \text{otherwise.} \end{cases}$$

For every $(j,k)_e \in A(D')$, let

$$\beta_{j,k} = \begin{cases} 1 & \text{if } P^j \cap P^k = \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

For every $(j,k)_{e_1}, (l,m)_{e_2} \in A(D')$, let

$$\gamma_{(j,k)_{e_1},(l,m)_{e_2}} = \begin{cases} 1 & \text{if } P^j, P^k, P^{\{j,k\}_{e_1}}, P^l, P^m, P^{\{l,m\}_{e_2}} \text{ are mutually disjoint} \\ 0 & \text{otherwise.} \end{cases}$$

Recall that for every node $i \in N(D)$, we created nodes $i, i' \in N(D')$. Consider such a pair $i, i' \in N(D')$. For the pair of nodes $i, i' \in N(D')$, we formulate the following problem:

$$\begin{aligned} Z_{(i,i')} = \quad & \text{Minimize} \sum_{(j,k)_e \in A(D')} w_{(j,k)_e} x_{(j,k)_e} \\ & \text{subject to} \\ & \qquad \sum_{(k,j)_e \in A(D')} x_{(k,j)_e} - \sum_{(j,k)_e \in A(D')} x_{(j,k)_e} = \alpha_j \quad \forall j \in N(D') \\ & \qquad x_{(j,k)_e} = 0 \quad \forall (j,k)_e \in A(D') : \beta_{j,k} = 0 \\ & \qquad x_{(j,k)_{e_1}} + x_{(l,m)_{e_2}} \leq 1 \quad \forall (j,k)_{e_1}, (l,m)_{e_2} \in A(D') : \gamma_{(j,k)_{e_1},(l,m)_{e_2}} = 0 \\ & \qquad x_{(j,k)_e} \in \{0,1\} \quad \forall (j,k)_e \in A(D') \end{aligned} \tag{23}$$

The solution to (23) represents a minimum weight (shortest) path on D'. The objective function contains the coefficients necessary to select the minimum weight path so that it would represent the most violated odd cycle on the hypergraph. The first constraint set enforces the continuity of the path from i to i'. The second constraint set ensures that the path does not contain any arcs such that the associated end partitions are not mutually disjoint. The third constraint set ensures that the path does not contain any pair of arcs such that the corresponding end and middle partition are not mutually disjoint. Finally, the fourth constraint set enforces the binary condition of the decision variables.

The path obtained by solving (23) can be used to construct a valid odd cycle on D, which in turn can be used to construct a valid odd cycle on G, from which one can obtain an odd cycle on H.

Observe that the original odd cycle separation algorithm for a directed graph D entailed solution of at most $N(D)$ shortest path problems. Therefore, it was polynomial in the size of the directed graph. However, the shortest path problem for conflict hypergraph carries additional constraints for ensuring mutual disjointness of the partitions. In addition, a conflict hypergraph might contain a multitude of edges intersections, thereby considerably increasing the problem data. This makes it much harder to implement to implement the separation routine for hypergraphs.

2.8 Concluding Remarks

In the preceding sections, we laid down a theoretical framework for obtaining facets of the independent set polytope from conflict hypergraphs. We provided generalized definitions for odd holes, odd antiholes, cliques, webs and antiwebs. Results for maximum vertex packing on hypergraph structures were derived. These yielded valid inequalities, which under certain conditions, were shown to be facet-defining for the entire independent set polytope. Later, the definitions were generalized further to include non-separable structures, and valid inequalities and facet-defining conditions were derived for them. We also derived Chvatal-Gomory ranks for the odd hole and clique inequalities. Finally, we provided a framework for creating separation routines for conflict hypergraphs, and illustrated the concept by extending the odd cycle separation algorithm.

However, some open questions remain. It would be interesting to investigate facet-defining conditions for valid inequalities derived from non-separable odd antiholes, webs and antiwebs. Currently, we have been able to get results only for cliques and odd holes. Another interesting question concerns the Chvatal-Gomory ranks for structures other than cliques and odd holes. It is our conjecture, due in large part to the correspondence between hypergraph structures and the associated reduced graphs, that Chvatal-Gomory ranks for other hypergraph inequalities should be same as those of the corresponding structures. It would also be interesting to investigate the theoretical complexity results for the separation problems for hypergraph structures. In fact, much more research is needed for creating practical separation routines that would be effective in real-world branch-and-cut algorithms.

CHAPTER III

COMPUTATIONAL EXPERIMENTS

In this chapter, we present computational experiments with hypergraph cutting planes. We first describe the problem instances. We explain their structure and discuss the factors that make them difficult to solve. We then describe implementations with hypergraph cutting planes, and present the computational results. We experiment with serial as well as parallel implementations.

3.1 Problem Instances

We chose a suite of multiple knapsack problems taken from JE Beasley's OR Library web-site [7]. These instances were developed by Chu and Beasley [14, 7].

The structure of these problem is of the form $max\{\sum_{j\in N} c_j x_j : \sum_{j\in N} a_{ij}x_j \leq b_i \forall i \in M\}$, where N, M are sets of columns and rows, respectively, and $x_j \in \{0,1\}$ for all $j \in N$.

For our computational experiments, we chose 30 instances with 250 variables and 10 constraints each. In these instances, each a_{ij} is a uniformly, randomly chosen integer between 0 and 1000. Each $b_i = \sum_{j\in N} a_{ij} \times \alpha$ for all $i \in M$, where $\alpha = 0.25$ for the first 10 problems, 0.5 for the next 10 problems, and 0.75 for the final 10 problems. The objective coefficients are computed as $c_j = \sum_{i\in M} \frac{a_{ij}}{|M|} + (500 \times p_j)$ where p_j is a uniform $(0,1)$ random variable. Thus, the objective coefficient for a column is obtained by summing the coefficients of that column over all the rows, and offsetting that column by a positive amount (this offsetting is necessary to avoid degeneracy).

These problems are especially suited to test the hypergraph cutting planes because of the following reasons. First, the tightness ratios ensure that it is highly unlikely that there exists a pair of binary variables that are in conflict with each other. Therefore, the use of conflict graphs is precluded. Second, because of the 100% dense nature of the problems, there exists a huge number of covers. Therefore, traditional lifted cover inequalities are not very effective. This is aptly demonstrated when CPLEX tries to solve these problems and generates thousands of cover inequalities in the process.

Problems of this nature have been discussed in [16, 1]. In literature, they are often referred to as *market-share* or *market-split* problems. These problems are particularly difficult to solve due to their dense and random nature. The effect of the density can be especially observed when CPLEX tries to generate its own cuts. Since every coefficient is a non-zero, CPLEX ends up generating hundreds of covers. The randomness is also quite problematic as it induces symmetry, leading to an explosion of the branch-and-bound nodes. Moreover, the uniform random distribution of the coefficients creates a constraint matrix that is *approximately* (as opposed to *exactly*) symmetric, which makes it difficult to employ symmetry breaking techniques. This effect can be seen in branch-and-bound progression, wherein the gap between upper and lower bounds tends to vary very little across the tree, and changes very slowly over time.

3.2 Implementation of Hypergraph Cuts

We experimented with a heuristic that generated odd hole and clique cuts. We chose these cuts because we were able to identify facet-defining conditions for the non-separable case. Moreover, there is considerable past research attesting to the efficacy of graph odd holes and cliques. Therefore, they provide a good starting point for testing hypergraph cuts.

3.2.1 Constrained Cover Generation

Here we describe a heuristic method that generates covers that include and exclude specified variables. Let N be the set of variables. Given a solution x^* of the current LP-relaxation, let $N_1 \subset N$ be the set of variables that must be included in the cover, and let $N_2 \subset N$ be the set of variables that must be excluded from the cover. For a given $i \in M$, the constrained cover generation heuristic attempts to solve the following optimization problem:

$$\begin{aligned}
\text{minimize } & \sum_{j \in N} (1 - x_j^*) z_j \\
\text{subject to } & \sum_{j \in N} a_{ij} z_j > b_i \\
& z_j = 1 \quad \forall j \in N_1 \\
& z_j = 0 \quad \forall j \in N_2 \\
& z_j \in \{0, 1\} \quad \forall j \in N
\end{aligned}$$

Clearly, a feasible solution to this problem represents a cover. The heuristic begins by selecting all the variables in the set N_1. The remaining variables are then sorted in the non-decreasing order of $\frac{1-x_j^*}{a_{ij}}$, $j \in N \setminus N_1$. The heuristic then picks up rest of the variables in this order until a cover is obtained. The heuristic is applied to every row $i \in M$, and the cover with maximum violation is selected.

3.2.2 Recursive Clique Generation

We generate cliques recursively using constrained cover generation in the following manner. Let S be a clique with partitions $P^i, P^{i,j}$, $i, j = 1, \ldots, d$, $i \neq j$. Let $V(S)$ be the set of variables currently in S. We *increment* S in the following manner. Generate an edge e_1 such that it includes at least one variable in P^1 and excludes all variables in $V(S) \setminus P^1$. Generate another edge e_2 such that it includes at least one variable in P^2, includes at least one variable in $e_1 \setminus V(S)$, and excludes all the variable in $V(S) \setminus P^2$. Observe that $e_1 \cap e_1 \neq \emptyset$. Finally, for $j = 3, \ldots, d$, generate edge e_j such that it includes at least one variable in P_j, includes at least one variable in $\cup_{k=1}^{j-1} e_k$, and excludes all variables in $V(S) \setminus P^j$. Thus, the original clique S can be recursively grown starting from an odd hole of size 3.

3.2.3 Recursive Odd Hole Generation

We generate odd holes recursively using constrained cover generation in the following manner. Let S be an odd hole with partitions $P^1, P^{1,2}, P^2, \ldots, P^{d-1}, P^{d-1,d}, P^d, P^{d,1}$. Let $V(S)$ be the set of variables currently in S. We *increment* S in the following manner. Generate an edge e_1 such that it includes at least one variable from P^1, includes at least one variable from $N \backslash V(S)$, and excludes all the variables in $V(S) \backslash P^1$. Next, generate an edge e_2 such that it includes at least one variable from P^2, excludes all the variables in $V(S) \setminus P^2$, and excludes all the variables in e_1. Finally generate an edge e_{12} such that in includes at least one variable in $e_1 \setminus V(S)$, includes at least one variable in $e_2 \setminus V(S)$, and excludes all the variables in $V(S)$. Remove the edge represented by $\{P^1 \cup P^{1,2} \cup P^2\}$ from the odd hole, and append the edges e_1, e_{12}, e_2. Thus, we have incremented the odd hole. This way, bigger odd holes can be recursively obtained starting from smaller ones.

3.2.4 Branch-and-Cut Implementation

We employed the hypergraph odd hole and clique cuts in a branch-and-cut scheme. The cut generation routine was initialized by generating an edge (cover) e_1 without any constraints. Then another edge e_2 was generated such that it included at least one variable in e_1, included at least one variable in $N \setminus e_1$, and excluded at least one variable in e_1. This creates an *edge-pair* with two intersecting edges. Then an attempt was made to generate a third edge e_3 such that it included at least on variable from $e_1 \setminus e_2$, and excluded all variables in $e_1 \cap e_2$. If this edge intersected both e_1 and e_2, then we have a clique of size 3. This clique was then grown recursively as described earlier.

On the other hand, if e_3 did not intersect e_2, then a fourth edge e_4 was generated such that it it included at least one variable in $e_3 \setminus e_1$, and excluded all variables in $e_1 \cup e_2$. Finally, a fifth edge e_5 was generated such that it included at least one variable in $e_4 \setminus e_3$, included at least one variable in $e_2 \setminus e_1$ and excluded all variables in edges $\{e_1 \cup e_3\}$. Thus we have an odd hole of size 5. this odd hole was then grown recursively as described earlier.

The entire process is illustrated in Figure 11. Each elipse in the illustration represents a cover or an edge. The illustration depicts the how the heuristic progressively builds the odd hole or clique structure.

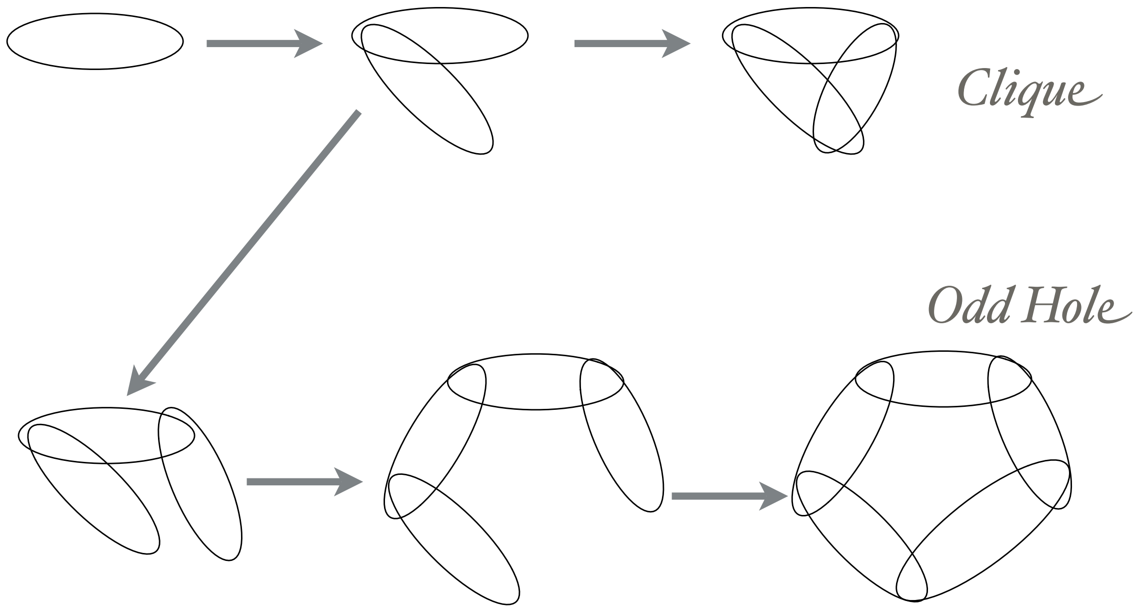


Figure 11: Clique and Odd Hole Cut Generation.

CPLEX 9 was used for the branch-and-bound implementation. All presolve features and CPLEX cuts were turned off for comparison. One round of cut generation was performed every 100000 branch-and-bound nodes. Maximum number of cuts added were limited to 10 (the number of rows in the constraint matrix). Lifting was performed for every cut and global validity of the cuts was maintained. The maximum limit on the size of the branch-and-bound tree was set at 6000 MB.

3.2.5 Computational Results

Our tests revealed that the current implementation of hypergraph odd hole and clique cuts is not very effective at solving the instances in our test suite. This is due to the following reasons. The effectiveness of the cuts rests heavily on the cover generation method, and the subsequent recursion. Being a heuristic, all the edges of the clique or odd hole generated might not be minimal. Therefore, the dimensionality of the cuts might be lower. Another reason is that the inherent nature of instances is such that the typical cover size is very large, primarily due to the tightness ratios described in Section 3.1. This leads to a lot of variables lying in the middle partitions, thereby increasing the right hand side of the valid inequalities and rendering them quite loose.

3.3 Implementation of Uniform Hypercliques

The computational experience with hypergraph odd holes and cliques led us to experiment with other types of valid inequalities derived from conflict hypergraphs. In particular, we decided to experiment with uniform hypercliques. These experiments are described below.

3.3.1 Uniform Hypercliques

In this section we summarize some of the important results related to uniform conflict hypergraphs and uniform hypercliques from Easton et al. [18]. Recall that a uniform k-hypergraph is a hypergraph such that all its edges have cardinality k. A hyperclique $K_{m,k}$ in a uniform k-hypergraph H_k is defined as a set of m vertices such that every $\binom{m}{k}$ combination of vertices comprises an edge. A hyperclique $K_{m,k}$ is said to be *maximal* if $\{v\} \cup K_{m,k}$ is not a hyperclique for all $v \in H_k \setminus K_{m,k}$.

Theorem 3.3.1. *[18] If $K_{m,k}$ is a hyperclique containing m vertices in a conflict k-hypergraph generated from an ISP instance, then $\sum_{i \in K_{m,k}} x_i \leq k-1$ is a valid inequality for P^{ISP}.*

Theorem 3.3.2. *[18] Let $K_{m,k}$ be a maximal hyperclique containing m elements in H_k with $n \geq m \geq k \geq 2$ (where n is the number of vertices in H_k). If for each vertex $v \in K_{m,k}$ there exists an edge $e \in E(K_{m,k})$ such that $v \in e$ and e is generated from a* *minimal dependent set, then* $\sum_{i \in K_{m,k}} x_i \leq k - 1$ *is facet-defining for* P^{ISP}.

Initial empirical tests revealed that the uniform hyperclique cuts by themselves are not very effective when the conventional branch-and-bound scheme is followed. Therefore, the original problem was partitioned into a number of sub-problems at the root node using a strategy similar to that mentioned in [18]. Each of these sub-problems was solved as an independent integer program with regular variable dichotomy, with the hypergraph cutting planes added to these sub-problems.

In the following sections, we describe the details of the computational strategy.

3.3.1.1 Aggregate Branching

The original problem was partitioned into sub-problems in the following manner. Let $S \subset N$ be a subset of the original variables. The first sub-problem was created by fixing $x_j = 1$ for all $j \in S$. A second set of $|S|$ sub-problems was created by setting $x_j = 1$ for all $j \in S \setminus \{k\}$, and $x_k = 0$ for all $k \in S$, $j \neq k$. A third set of $\frac{|S| \times (|S|-1)}{2}$ sub-problems was created by fixing $x_j = 1$ for all $j \in S \setminus \{k, l\}$, and $x_k = x_l = 0$ for all all $k, l \in S$, $j \neq k \neq l$. Finally, a fourth sub-problem was created by appending the inequality $\sum_{j \in S} x_j \leq |S| - 3$ to the original instance. Therefore, this scheme partitions the original problem into $(1 + |S| + \frac{|S| \times (|S|-1)}{2} + 1)$ sub-problems. Each of these sub-problems is then solved as an independent integer program. This partitioning process called *aggregate branching*, is illustrated in the Figure 12. Note that aggregate branching is performed at the root node only. Thereafter, each of the sub-problems is solved independently using regular variable dichotomy.

For our experiments, the construction of subset S was guided by the solution of the LP-relaxation of the original problem. Let r be reduced costs of the primal variables. We chose $S = \{j : r_j \geq \epsilon\}$. The value of ϵ was chosen such that $|S| \approx \rho \times |N|$. We chose parameter $\rho = 0.2$.

The aggregate branching strategy helps in a number of ways. First, if the set

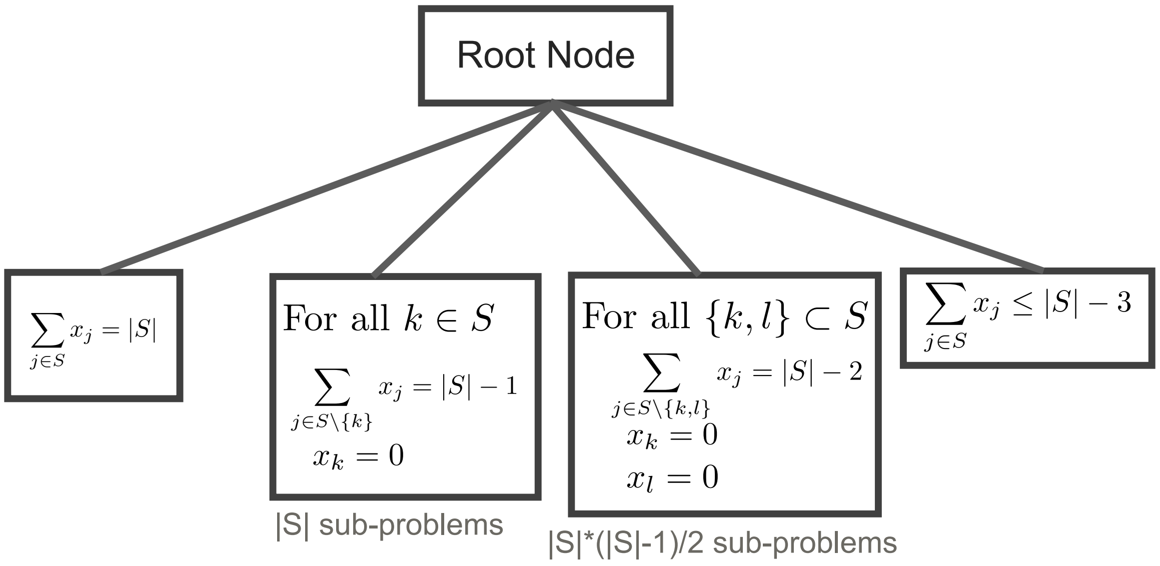

Root Node
$\sum_{j\in S} x_j = |S|$
For all $k \in S$
$\sum_{j\in S\setminus\{k\}} x_j = |S|-1$
$x_k = 0$
|S| sub-problems
For all $\{k,l\} \subset S$
$\sum_{j\in S\setminus\{k,l\}} x_j = |S|-2$
$x_k = 0$
$x_l = 0$
|S|*(|S|-1)/2 sub-problems
$\sum_{j\in S} x_j \le |S|-3$

Figure 12: Aggregate Branching.

S is chosen such that it contains a lot of the variable that would have value 1 in the optimal solution, then we can get good lower bounds early in the optimization. This helps in pruning a lot of branch-and-bound nodes. This is critical since these instances tend to exhaust all branch-and-bound nodes before proven-optimality is reached (e.g., in CPLEX runnings, it is typical to see 20 million branch-and-bound nodes being generated.) Second, because so many of the variables are fixed, the dimensionality of the resulting sub-problem is reduced, thereby aiding the process of growing the hyperclique (see Section 3.3.1.2). Finally, because a large proportion of the variables is fixed, it reduces the symmetry of the problems. All these factors together help in speeding up the solution times.

3.3.1.2 Generating Uniform Hyperclique Cuts

The uniform hyperclique cuts were generated in the following manner. Let $a_j = \sum_{i\in M} a_{ij}$, $j \in N$. Let $b = \sum_{i\in M} b_i$. We sum up the coefficients like this because we found that the summed constraint provides a better relaxation to the original problem by reducing much of the randomness in the constraint matrix.

Without loss of generality, let $a_1 \geq a_2 \geq \ldots \geq a_{|N|}$. Let k be a fixed integer. The uniform hyperclique can be constructed by finding the largest q such that $\sum_{j=q-k+1}^{q} a_j > b$. Clearly, all subsets of size k of the set $K_{q,k} = \{1, \ldots, q\}$ form a dependent set. Therefore $K_{q,k}$ represents a uniform k-hyperclique of size q, and the inequality $\sum_{j \in K_{q,k}} x_j \leq k - 1$ is valid.

In the interest of stronger valid inequalities, we would like to get as large a hyperclique as possible. Therefore, we employed a strategy that iteratively *grows* the hyperclique obtained above. This strategy, which is similar to sequential lifting, is as follows.

For any $l \in N \setminus K_{q,k}$, consider the following integer program:

$$\begin{aligned} \text{maximize} \ & \sum_{j \in K_{q,k}} x_j \\ \text{subject to} \ & \sum_{j \in N} a_{ij} x_j \leq b_i \quad \forall i \in M \\ & x_l = 1 \\ & x_j \in \{0, 1\} \quad \forall j \in N \end{aligned} \tag{24}$$

If the optimal value of the integer program (24) is $k - 2$ or less, then l can be added to $K_{q,k}$. Clearly, it is difficult to solve this optimization problem repeatedly for different l. However, solution time may be controlled by employing some heuristic strategies. First, because k is known beforehand, we can set the cut-off value at $k-1$. Also, one can set a parameter δ such that the optimization is halted if the solution of LP-relaxation exceeds δ. Finally, a limit can be set on the maximum number of branch-and-bound nodes processed during the optimization.

Since the process of generating uniform hyperclique inequalities is computationally expensive, it is not feasible to generate them frequently (as in a branch-and-cut scheme). Initial empirical tests showed that first sub-problem (where all the elements of $S \subset N$ were fixed to 1), and the last sub-problem (where none of the variables were fixed and an inequality was added), took longer than the others. Therefore, hyperclique cuts were generated for these two sub-problems only.

Specifically, two hyperclique inequalities were added to each of the aforementioned sub-problems. The choice of k for both the inequalities was guided by the solution of the initial LP relaxation. For the first hyperclique inequality, we set $k = \lfloor \sum_{j \in N} x_j^* \rfloor$, and for the second hyperclique, we set $k = \lfloor \sum_{j \in N} x_j^* \rfloor - 1$.

3.3.2 Results

The tests reported here were run on a Linux cluster with eight quad-core 2.33 GHz Intel(R) Xeon(R) processors, and 12 GB of RAM. The experiments were performed using CPLEX 9 as an LP solver.

In this section, we describe the results from the serial implementation running on a single processor (see Section 3.4 for the parallel implementation). For comparison, we benchmarked against CPLEX running time. We set a limit of 6000 MB on the size of the branch-and-bound tree. We found that the CPLEX performed best with the MIP emphasis parameter set to *best bound*. For the purpose of comparison, CPLEX cuts were turned off when implementing uniform hypercliques and aggregate branching.

The results are summarized in the table 1.

Table 1: Comparison of Uniform Hyperclique Cuts and Aggregate Branching with CPLEX

Instance	Optimal IP Obj.	Time (CPU seconds)				
		CPLEX	**Uni HypClq + Agg Branch**			
			Total	**1st SubProb**	**Last SubProb**	**Cut Gen**
mk5-01	59187	27337	16846	3914	1492	627
mk5-03	58097	8395	2828	1009	70	535
mk5-06	58824	7984	3572	1099	129	543
mk5-07	58704	6399	2780	448	823	631
mk5-08	58936	CPXERR	108522	5668	17371	464
mk5-09	59387	23015	6547	1031	1830	604
mk5-10	59208	39706	14213	1442	5549	494
mk5-11	110913	27204	20411	20077	0	533
mk5-16	110845	CPXERR	40340	39981	0	504
mk5-18	106686	40458	27389	26981	0	615

The first column of the table contains the names of the instances. The second column contains the optimal IP objective value. The third column contains the time taken by CPLEX to solve the instance to optimality. All times reported in this table are in CPU seconds. The entry "CPXERR" indicates that the limit on the branch-and-bound tree was reached. The fourth column total time taken to solve the instances to optimality with uniform hypercliques and aggregate branching strategy. The fifth column contains the times taken by the first aggregate branching sub-problem. The sixth column contains the times taken by the last aggregate branching sub-problem. Finally, the last column contains the time spent in generating the uniform hyperclique cuts (recall that two cuts were generated for the first sub problem, and two for the last sub problem).

The tests are reported for ten instances. Our implementation with uniform hyperclique cuts and aggregate branching produces much better solution times than CPLEX. All implementations were solved to proven optimality by our implementation of uniform hyperclique cuts. CPLEX, however, was able to solve only eight of these ten instances. CPLEX exceeded the tree limit memory for the two instances, namely mk5-08, and mk5-16. Hence, branch-and-bound was terminated.

The total computational time varied widely. The minimum time taken by CPLEX was 6399 CPU seconds for instance mk5-07. Among other instances that CPLEX was able to solve, it took the longest for mk5-18 (40458 CPU seconds).

Our implementation of uniform hyperclique cuts combined with aggregated branching decreases the solution times significantly. For the instances mk5-08 and mk516 (for which CPLEX ran out of memory), our implementation took 108522 and 40458 CPU seconds, respectively. For instance mk5-09, CPLEX took 23015 CPU seconds, while our implementation took 6547 CPU seconds. For instance mk5-03, our implementation took 2828 CPU seconds, against the 8395 CPU seconds taken by CPLEX.

For instance mk5-07 (for which CPLEX performed the best, taking 6399 CPU seconds), our implementation took 2780 CPU seconds. Our implementation took the most amount of time for instance mk5-08, and the least amount of time for instance mk5-07.

The total cut generation time for our implementation was fairly uniform, varying from 464 CPU seconds for mk5-08 to 631 CPU seconds for mk5-07. Interestingly, our implementation performed best for mk5-07 and worst for mk5-08.

It is interesting to compare the solution time for the first and the last aggregate branching sub-problems. Intuitively, the first sub-problem should take less time (because the maximum number of variables were fixed in it), and the last sub-problem should take longer (because no variables were fixed). This is indeed the case for instances mk5-07, mk5-08, mk5-09, and mk5-10.

However, the reverse is true for other instances. For example, in case of instance mk5-01, the first sub-problem took 1009 CPU seconds, while the last sub-problem took 70 CPU seconds only. The time for the first sub-problems for instances mk5-11, mk5-16 and mk6-18 were 20077 CPU seconds, 39981 CPU seconds and 26981 CPU seconds, respectively. On the other hand, the last sub-problem took merely a fraction of a second for each of these instances. In case of these instances, the first sub-problem contained a lot of the columns that would take value 1 in the optimal solution. This generated strong feasible solutions early on in the tree. Moreover, the nature of the instances is such that in case of last sub-problem, nodes were heavily pruned due to reduced cost fixing. In fact, in case of instances mk5-11, mk5-16 and mk6-18, all the columns could be fixed at the (subproblem)root node itself.

It is also interesting to observe that even though the coefficients are uniformly randomly distributed, there is such a wide disparity in the solution times with CPLEX as well as uniform hypercliques with aggregate branching.

Another important observation relates to the tightness ratio of the instances.

Recall that the right hand sides of the knapsack constraints are generated by simply adding up the coefficients and using a multiplier to reduce the sum. This multiplier is referred to as the tightness ratio. There are three tightness ratios used: 0.25, 0.50, and 0.75. The first seven instances reported in the table have a tightness ratio of 0.25. CPLEX performs particularly poorly in these instances. On the other hand, the rest of the instances have a tightness ratio of 0.50. In these last three instances, CPLEX performance is much more comparable with our implementation.

This observation can be explained by considering a traditional knapsack problem. If the size of the knapsack is small, then there might be a lot of different combinations that can give a high objective value. However, when the knapsack capacity is large, most of the valuable items can be easily put in. The resulting number of combinations due to the remaining low value items is not that high. In other words, one need not consider as many possible choices with a large knapsack as one might need to consider with a smaller knapsack. The same principle can be observed here. With a larger tightness ratio, most of the columns with high objective coefficients can be easily set to one. This leads to a smaller gap between the LP relaxation and the feasible solution. It also reduces the number of good combinations remaining, which leads to large number of nodes getting pruned.

3.4 Parallel Implementation

In this section, we describe a parallel implementation of computational experiments. We first describe the motivating factors that led us to consider parallel computing. We then describe the parallel implementation itself, and the issues involved therein. Finally, we report the results of our experiments.

3.4.1 Motivation

There are a number of reasons that motivate a parallel implementation of the computational experiments described in the previous sections.

Observe that the serial computational strategy described relies heavily on partitioning of the problem (through aggregate branching). Moreover, the branch-and-bound process itself creates a number of sub-problems, each of which can be solved separately. In fact, the nature of our problem instances is such that the number of branch-and-bound nodes processed is extremely large. These factors make the problem particularly suited for parallel computing.

The issue of cutting planes is another factor that motivates a parallel implementation. In regular parallel branch-and-cut algorithms, if the number of cuts generated is large, then the transfer of cuts from one processor to another can become a big issue, adding significantly to communication time. However, our single-processor computational strategy involves very few cutting planes. Therefore, this additional communication overhead among processors may be avoided by generating the cuts locally.

Another issue in parallel branch-and-bound involves the branching process which uses the dual-basis resident in the parent node to initialize the dual-simplex algorithm for solving the children. If the LP relaxation takes a large number of iterations to converge, then one needs to some how transfer the resident basis along with the branch-and-bound nodes. However, the LP solution time for our test-set is very low. Therefore, nodes can be transferred without the basis information, thereby incurring less communication overhead.

To summarize, our single-processor computational strategy involves aggressive partitioning and few cutting planes. Moreover, the nature of the problems is such that the LP relaxation solution times are very small, but the the number of branch-and-bound nodes is extremely high. Therefore, parallel implementation is a natural extension of the current work.

3.4.2 Previous Research

Literature concerning parallel branch-and-bound dates back to 1970's. Pruul et al. [39] report results of a simulated parallel branch and bound algorithm for solving traveling salesman problem. Gendron and Crainic [22, 23] provide a survey of work on parallel branch and bound. Ashford et al. [3] present a more sophisticated branch and bound algorithm for a transputer network of up to eight nodes. Eckstein was among the first to write an industrial strength parallel branch and bound code for general mixed integer programming. An implementation is discussed in [20]. In a later work, he discusses the possibility that knowledge of advanced features of computer architecture might not be necessary to write effective parallel solvers [19]. Cannon and Hoffman [13] were the first to report results of a parallel branch and cut implementation. Bixby et al. [9, 11] use a parallel software platform called TreadMarks [28] for implementing parallel branch-and-cut. There has also been some research on employing parallel computing for structured integer programs. [2] describe a parallel implementation for the traveling salesman problem. Linderoth [31] investigates parallel schemes for set partitioning problem.

3.4.3 Parallel Implementation Strategies

We used the Message Passing Interface in a distributed memory environment for our implementation. We followed a *master-slave* paradigm wherein the master processor does not perform any optimization. All optimization is done by the slave processors. Master only keeps track of global data (such as the global best feasible solution) and acts as an intermediary to facilitate communication among difference slaves.

3.4.3.1 Work Distribution

Initially, a queue of the aggregate branching sub-problems is kept by the master processor. It polls for work-requests from each of the slave processors. If a slave is idle, it sends a request for work to the master. When the master receives a request from a slave, it sends the next sub-problem from the queue. This strategy is followed until the master processor runs out of aggregate branching sub-problems. Thereafter, upon arrival of new work-requests, master queries a busy slave for some of its unsolved branch-and-bound nodes, and transfers those nodes to the idle slave. When the master determines that all the slaves are idle, it sends a signal for global exit, which terminates the program across all processors.

Whenever a busy slave receives a new request from the master, it allocates a buffer to store the nodes that will be sent over. Upon allocation of this buffer, every time this slave is about to branch on a node, it evaluates whether to add this node to the buffer or to process it locally. If the decision is made to add the node to the buffer, then no branches are created, and the node is added to the buffer. Otherwise, the node is processed as usual locally. The process continues until the requisite number of nodes have been added to the buffer. When this occurs, the buffer is sent to the master. The master, acting as an intermediary to facilitate communication, forwards this buffer to the idle slave that sent the original work request.

3.4.3.2 Cut Generation

The cut generation strategy for the parallel implementation is same as the one described in the serial implementation (see Section 3.3.1.2). Two hyperclique inequalities are added to the first aggregate branching sub-problem, and two are added to the last sub-problem. The inequalities are generated locally by the slave processors upon receiving the appropriate aggregate branching sub-problem from the master processor.

3.4.3.3 Lower Bounds

During the course of the optimization, new lower bounds (feasible solutions) are found by the slave processors. Whenever a slave discovers a new lower bound, it transmits it to the master processor. The master compares it with the currently stored value

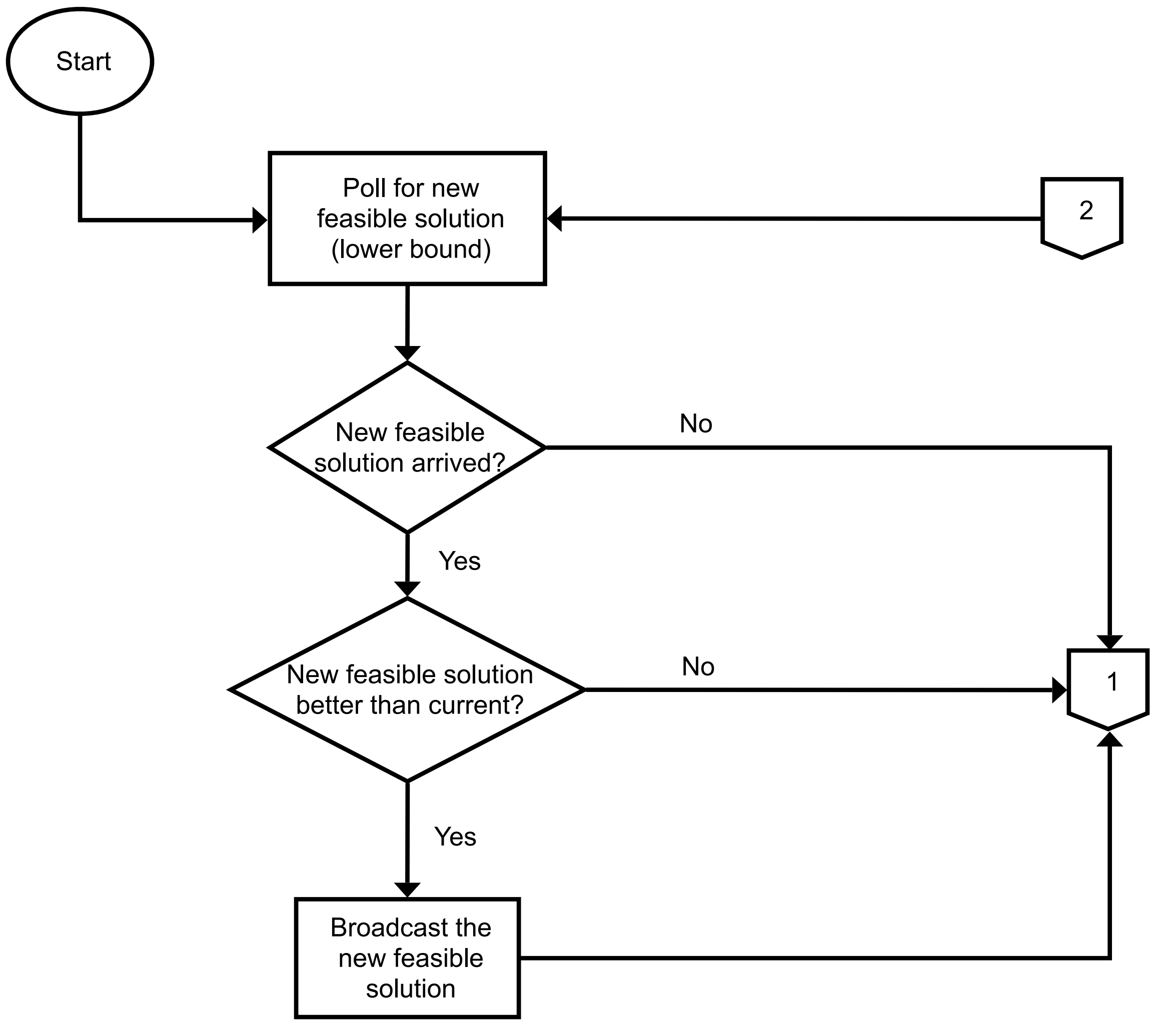

Start
Poll for new feasible solution (lower bound)
2
New feasible solution arrived?
No
Yes
New feasible solution better than current?
No
1
Yes
Broadcast the new feasible solution

Figure 13: Master - Module 1.

of the global lower bound. If the new value sent by the slave is greater than than the current lower bound, then the master updates the current stored value of the global lower bound and broadcasts it to all the slaves.

Figures 13 and 14 pictorially depict the implementation in the master processor. Figures 15 and 16 illustrate the implementation for the slave processors.

3.4.3.4 Work Distribution Issues and Trade-offs

When a busy slave receives requests for some of its nodes, some important decisions need to be made. The first of these decisions is related to the number of nodes to send. If too few nodes are sent, then there might be too many "request-send" transactions,

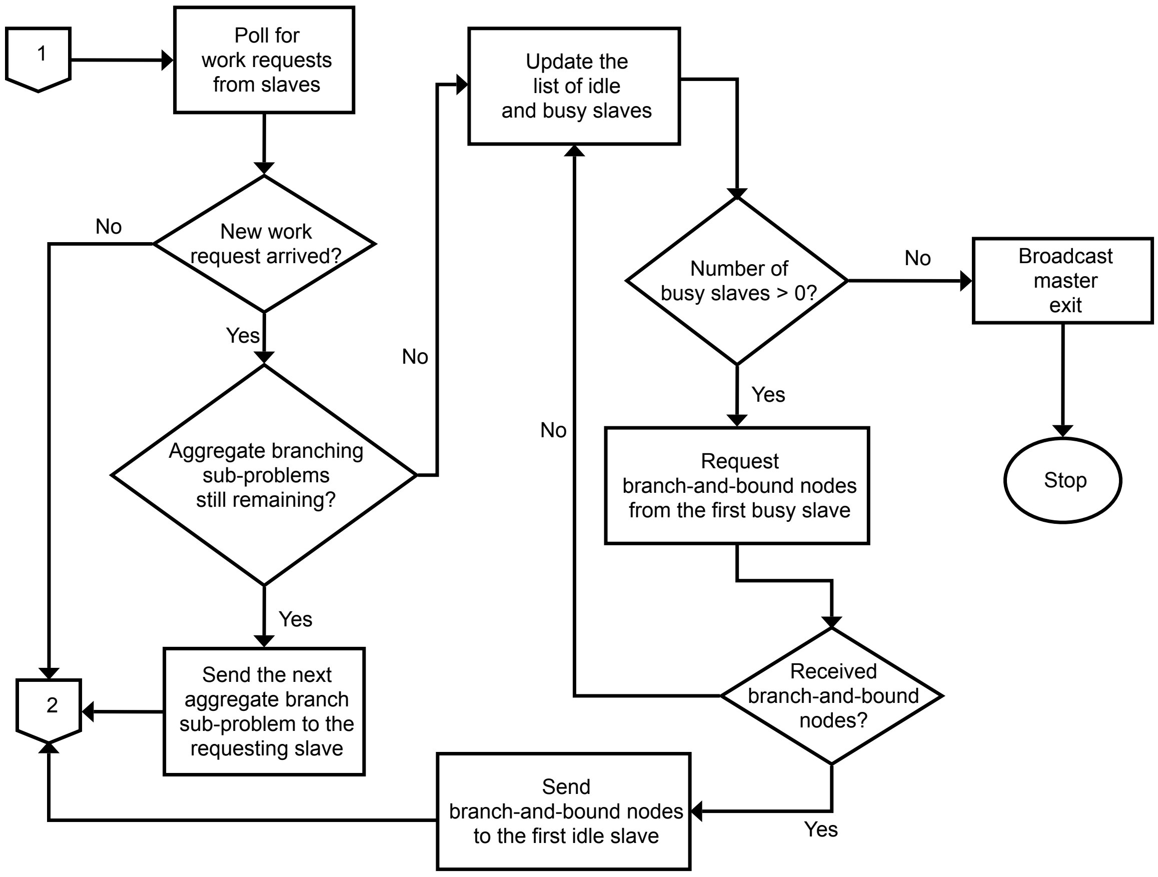

1
Poll for work requests from slaves
New work request arrived?
No
Yes
Aggregate branching sub-problems still remaining?
No
Yes
Send the next aggregate branch sub-problem to the requesting slave
2
Update the list of idle and busy slaves
Number of busy slaves > 0?
No
Broadcast master exit
Stop
Yes
Request branch-and-bound nodes from the first busy slave
Received branch-and-bound nodes?
No
Yes
Send branch-and-bound nodes to the first idle slave

Figure 14: Master - Module 2.

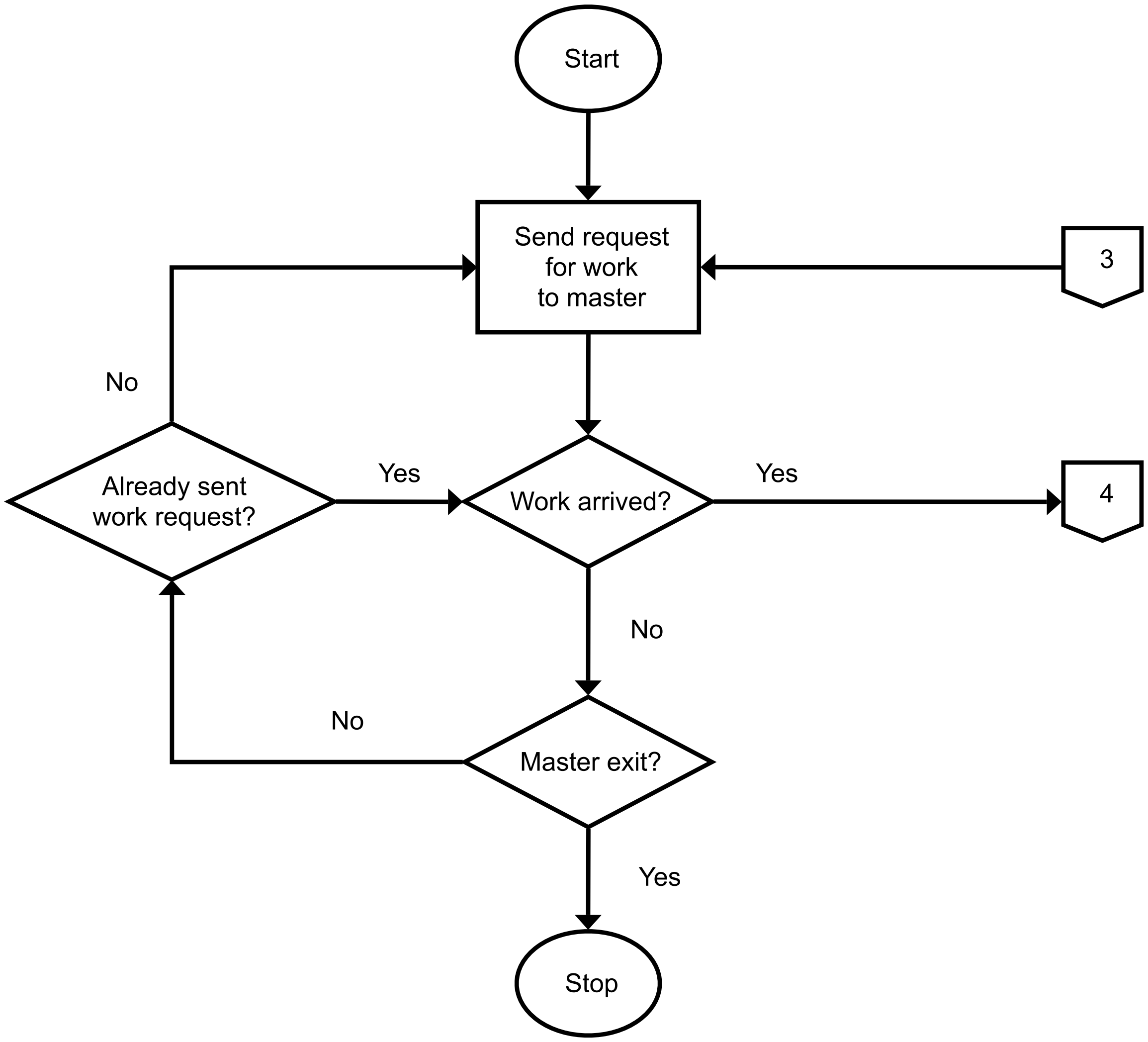


Figure 15: Slave - Module 1.

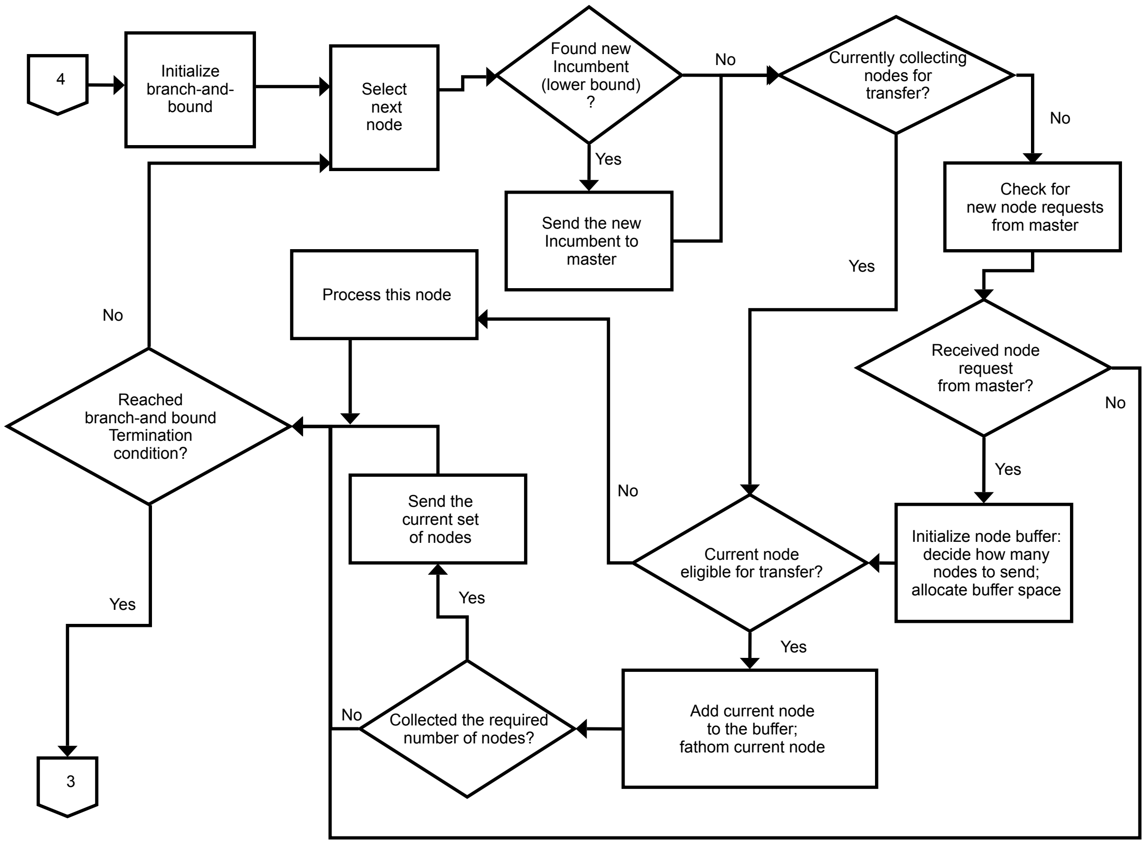

4
Initialize branch-and-bound
Select next node
Found new Incumbent (lower bound) ?
No
Yes
Send the new Incumbent to master
Currently collecting nodes for transfer?
No
Yes
Check for new node requests from master
Received node request from master?
No
Yes
Initialize node buffer: decide how many nodes to send; allocate buffer space
Process this node
Current node eligible for transfer?
No
Yes
Add current node to the buffer; fathom current node
Collected the required number of nodes?
No
Yes
Send the current set of nodes
Reached branch-and bound Termination condition?
No
Yes
3

Figure 16: Slave - Module 2.

adding to communication overhead. On the other hand if too many nodes are sent, the sender itself might become idle quite soon. In fact, if a large, fixed number of nodes are sent every time, there is a possibility of entering a cycle wherein two slaves exchange their sender and recipient status endlessly. Also, whenever communication between two processors takes place, a buffer has to be allocated on each of the processors. If the number of nodes transferred in a single transactions is very large, then the buffer needs to be sufficiently large too. If these buffers are excessively large, it may lead to memory problems.

One way to make this decision is to send a fraction of the nodes left. In the current implementation, we send $min(l, \frac{n}{s+1})$ nodes, where l is a pre-decided limit on the maximum number of nodes to send, n is the number of nodes left, and s is the number of idle slaves. We refer to l as the *node transfer limit.* This pre-decided limit is necessary because the receiving processor needs to allocate a sufficient buffer to store the incoming data.

Note that this node transfer limit imposes an upper limit on the total number of nodes transferred at one time. This is different from the total number of nodes transferred during the course of the optimization. For example, if the node transfer limit is 1000, it simply means that at most 1000 nodes may be transferred in a single transaction. Over the course of the optimization, a number of such transactions may occur, and the total number of nodes transferred may be much greater.

There also needs to be a minimum number of nodes left for the transfer to occur. If the number of nodes left is less than this minimum value, then the busy slave should not send any nodes. This limit is needed so that the busy slave does not run out of nodes by sending too many of its nodes. In the current implementation, we chose the node transfer limit itself as this minimum requirement. Thus, if the number of nodes left at a slave is less than this minimum requirement, then no nodes are sent.

Another decision is related to the selection of the nodes to send. Ideally, one would like to send nodes that are likely to generate large sub-trees. However, this is very difficult to predict for any given node. The gap between the node objective value and the current incumbent solution is one indicator that can be used (nodes large gaps are likely to large sub-trees, and vice versa).

In this implementation, we chose the *percentage gap* of a branch-and-bound node as an indicator for the size of the sub-tree generated by this node. We defined the percentage gap as $\frac{(ub-lb)\times 100}{lb}$, where ub is the node objective value, and lb is the current best lower bound. If this percentage gap is high, then the sub-tree is likely to be large too. We chose a minimum percentage gap requirement to decide whether a given branch-and-bound node should be transferred or not. If the percentage gap at a node is less than this minimum requirement, then that node is not transferred and processed locally instead.

Again, there are trade-off associated with the minimum required percentage gap. If this minimum requirement is too low, then nodes with small sub-trees might start getting transferred. This means that the slave receiving these nodes will quickly process them and become idle again. On the other hand if this requirement is too high, the busy slave might not be able to find too many nodes to send over, and might take a long time to build up the buffer of nodes to send over. As a result, the idle slave might have to wait a long time before it receives work. This will lead to excessive wait times.

3.4.4 Results

In this section, we report the results from our computational experiments with parallel implementation. We experimented with two values of the maximum node transfer limit (100 and 1000). As mentioned earlier, this refers to the limit on node transfer at a given time. It is different from the total number of nodes transferred throughout the running of the program. We also experimented with two values of the minimum percentage gap (0.1 and 0.0). Thus, there was no minimum percentage gap required for node transfer in the latter case.

The results of computational experiments from the parallel implementation are reported in the tables 2 through 7.

There is a separate table for each instance. In each table, the term *MinPctGap* refers to the parameter minimum percentage gap. Similarly, the term *NodeTransfer-Lim* refers to the parameter maximum node transfer limit.

Each table is divided into two halves. The top half reports results when the minimum percentage gap was set at 0.1 percent, and the bottom half reports results when this parameter was set at 0.0 percent. Each of these halves is again divided into two halves - left and right. The left half reports the results for the parameter maximum node transfer limit set at 100, and the right half reports the results maximum node transfer limit at 1000. Thus, each table is partitioned into four quadrants. Each quadrant contains five columns. The first of these column is labeled *Processors* and refers to the total number of processors used. The second column is labeled *Total Time* and reports the total solution time (in CPU seconds). The third column is labeled *Max Wait Time* and reports the maximum total waiting time (CPU seconds) among all the slave processors. A slave processor is considered to be waiting if it has sent a request for additional work to the master, but has not yet received any problems. Total waiting time of a slave refers to the total time it spent in waiting state. The fourth column labeled *Nodes Transferred* reports the total number of branch-and-bound nodes transferred from one slave to another (via the master acting as an intermediary). Finally, the the fifth column labeled *Speedup* reports the ratio of total solution time with multiple processors to the total solution time with a single processor. In all tables, the times with single processor are also reported for comparison.

Figures 17 through 23 plot the total time taken against the number of processors employed for different combinations of the parameters. The y-axis values represent the total CPU seconds taken, and the x-axis values represent the number of processors used.

The total times for all instances seem to show significant decrease up to five processors. The decrease is not significant thereafter. The maximum speedup obtained is 5.43 for instance mk5-10. In general, it is noticed that speed ups are much better when the solution time with a single processor is also high. Speed ups are not significant for cases where the single processor solution time is less. For example, the single processor solution time for instance mk5-03 is 2828 CPU seconds, and the maximum speedup obtained is merely 2.49.

It is interesting to note how the instances behave for different values of the parameters. The total number of nodes transferred increases significantly when the maximum node transfer limit is increased and the minimum percentage gap requirement is relaxed. However, this does not necessarily translate into better performance. This can be seen by observing the wait time values, which do not seem to follow the number of nodes transferred in a consistent manner. This can be attributed to the increased communication overhead encountered when a large number of nodes are transferred from one processor to another.

We also observed an anomaly. Normally, solution time decreases as the number of processors increase. This is the case even when speed ups are small. However, for instance mk5-09, the total solution time jumps significantly with seven processors when the minimum percentage gap requirement is 0.0 and maximum node transfer limit is 100. In this case, it was observed that most of the nodes transferred to the recipient idle slave were quickly processed (because minimum percentage gap was 0.0), and the slave would become idle almost immediately. As a result, the master would enter long cycles where it would repeatedly serve the request from this one slave, while other slave requests were neglected. This behavior of the system manifested in the unusually high maximum waiting time (4305.07 CPU seconds) and nodes transferred (693869 branch-and-bound nodes).

Clearly, the very nature of parallel computing makes it unpredictable, forcing any investigations to be of a largely empirical character. This is due to the multitude of factors influencing system performance, and the sensitivity of the system to even small changes in these parameters.

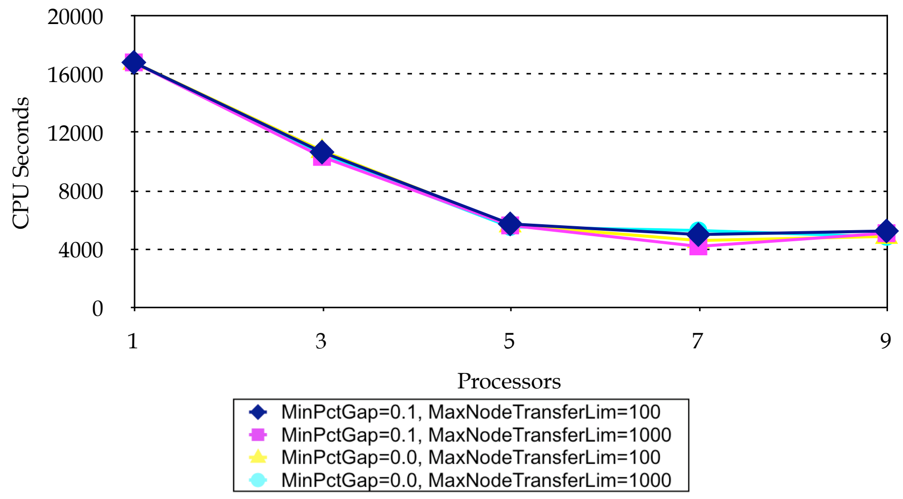


Figure 17: Total Time for Instance mk5-01.

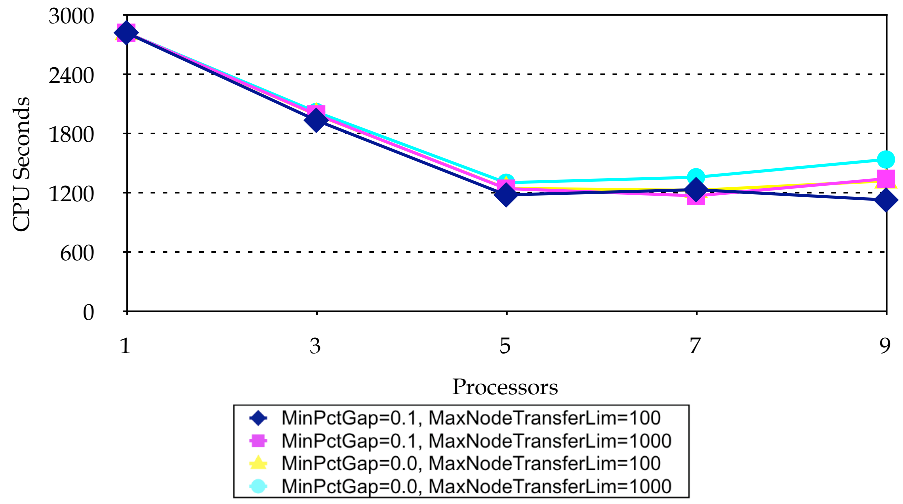


Figure 18: Total Time for Instance mk5-03.

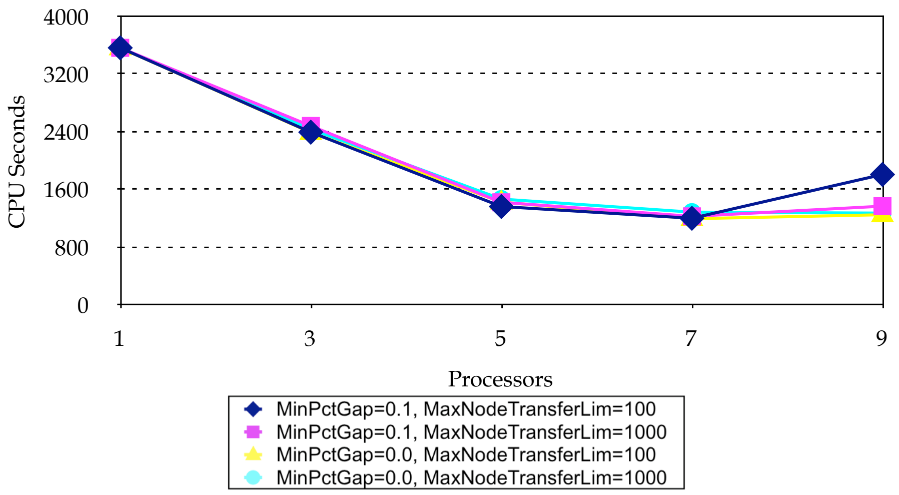


Figure 19: Total Time for Instance mk5-06.

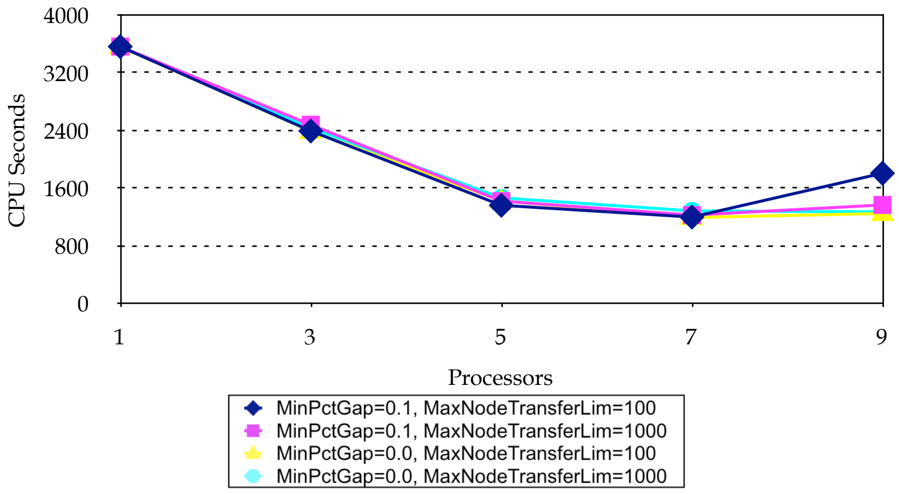


Figure 20: Total Time for Instance mk5-06.

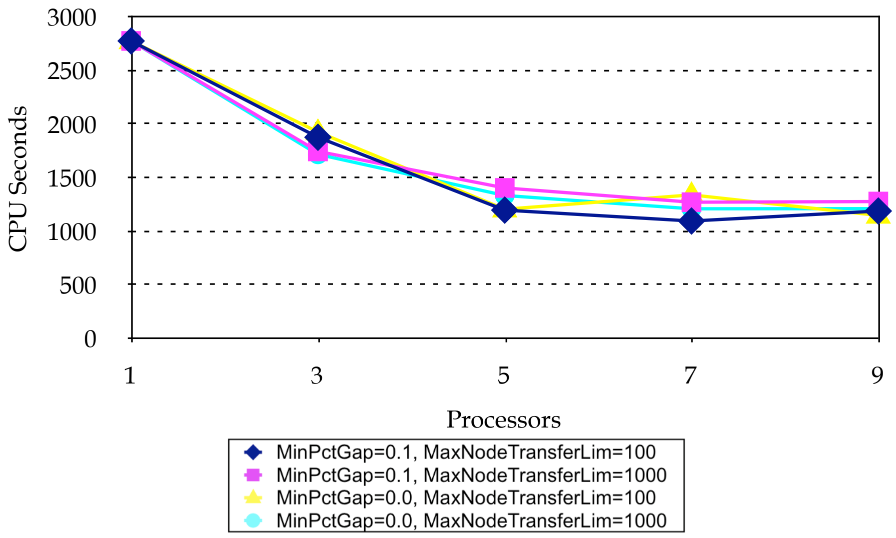


Figure 21: Total Time for Instance mk5-07.

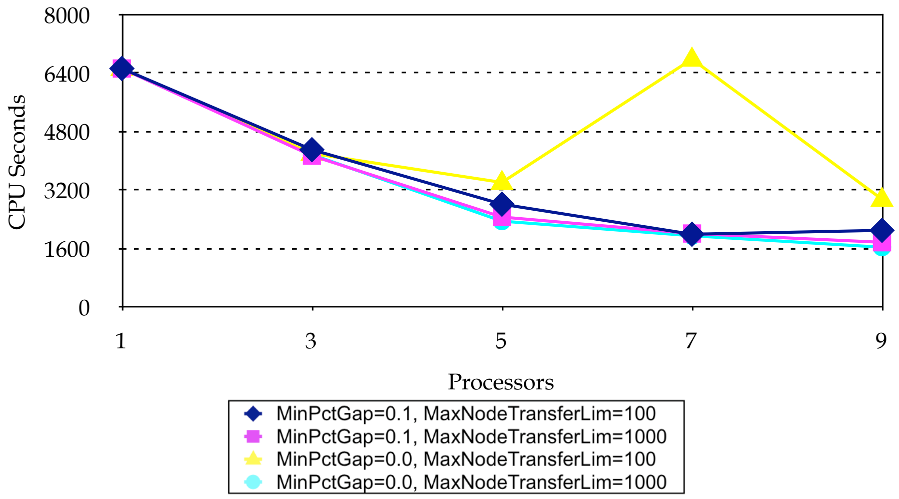


Figure 22: Total Time for Instance mk5-09.

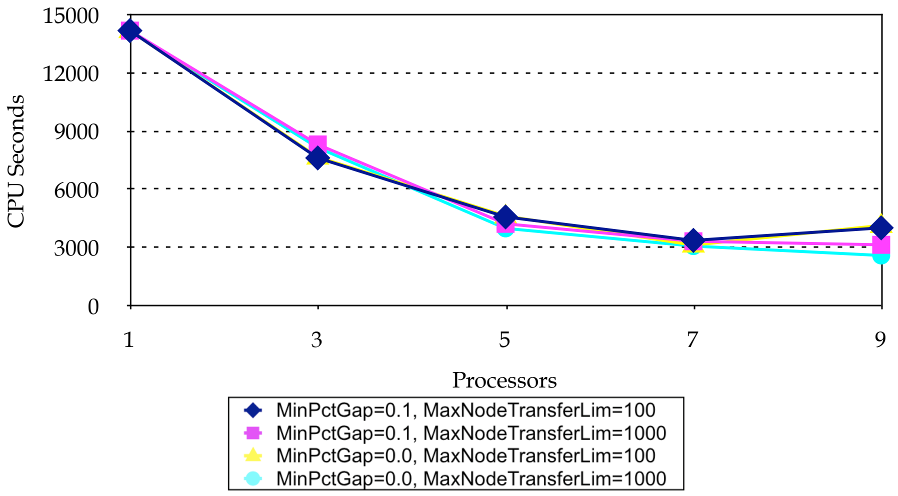


Figure 23: Total Time for Instance mk5-10.

Table 2: Parallel Implementation Results for Instance mk5-01

mk5-01									
MinPctGap = 0.1									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*
1	16846.00	-	-	-	1	16846.00	-	-	-
3	10699.40	452.44	33	1.57	3	10371.60	431.77	610	1.62
5	5779.18	684.96	440	2.91	5	5676.63	666.29	2203	2.97
7	5047.81	670.74	696	3.34	7	4239.29	454.15	1244	3.97
9	5293.66	699.28	813	3.18	9	5142.66	717.82	1800	3.28
MinPctGap = 0.0									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*
1	16846.00	-	-	-	1	16846.00	-	-	-
3	10802.90	573.29	69651	1.56	3	10506.20	488.16	10987	1.60
5	5708.84	505.13	50827	2.95	5	5573.15	493.12	15669	3.02
7	4641.26	496.48	802246	3.63	7	5327.85	721.85	827830	3.16
9	4921.13	873.60	1481046	3.42	9	4894.19	636.82	1467771	3.44

Note: All times are reported in CPU Seconds

Table 3: Parallel Implementation Results for Instance mk5-03

mk5-03									
MinPctGap = 0.1									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*
1	2828.00	-	-	-	1	2828.00	-	-	-
3	1940.91	359.02	33	1.46	3	2000.60	359.37	333	1.41
5	1185.51	376.93	182	2.39	5	1249.66	389.26	724	2.26
7	1238.80	387.93	33	2.28	7	1176.13	384.94	729	2.40
9	1134.26	375.52	6091	2.49	9	1349.23	296.41	10777	2.10
MinPctGap = 0.0									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*
1	2828.00	-	-	-	1	2828.00	-	-	-
3	1997.91	363.87	3096	1.42	3	2025.78	357.91	2290	1.40
5	1252.21	130.96	188300	2.26	5	1309.08	88.92	144400	2.16
7	1232.33	210.05	293860	2.29	7	1364.77	205.58	305362	2.07
9	1327.24	312.39	436232	2.13	9	1544.20	391.56	428802	1.83

Note: All times are reported in CPU Seconds

Table 4: Parallel Implementation Results for Instance mk5-06

mk5-06									
MinPctGap = 0.1									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*
1	3572.00	-	-	-	1	3572.00	-	-	-
3	2400.99	380.64	60	1.49	3	2487.51	386.61	333	1.44
5	1369.51	361.75	229	2.61	5	1428.02	361.50	702	2.50
7	1206.41	369.73	33	2.96	7	1233.09	399.86	726	2.90
9	1814.03	383.74	33	1.97	9	1374.38	364.12	706	2.60
MinPctGap = 0.0									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*
1	3572.00	-	-	-	1	3572.00	-	-	-
3	2397.26	362.93	133	1.49	3	2443.77	360.07	1677	1.46
5	1431.16	199.05	137841	2.50	5	1473.00	161.39	107247	2.42
7	1200.23	90.02	337756	2.98	7	1291.82	81.42	277000	2.77
9	1255.52	96.61	495612	2.85	9	1278.30	93.58	416000	2.79

Note: All times are reported in CPU Seconds

Table 5: Parallel Implementation Results for Instance mk5-07

mk5-07									
MinPctGap = 0.1									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*
1	2780.00	-	-	-	1	2780.00	-	-	-
3	1881.61	384.03	33	1.48	3	1749.90	389.54	333	1.59
5	1202.27	469.98	528	2.31	5	1409.11	458.39	916	1.97
7	1100.73	537.97	554	2.53	7	1276.12	473.50	1949	2.18
9	1193.97	541.23	167	2.33	9	1282.28	511.42	2958	2.17
MinPctGap = 0.0									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*
1	2780.00	-	-	-	1	2780.00	-	-	-
3	1931.33	470.09	44548	1.44	3	1722.48	384.73	2333	1.61
5	1210.55	397.36	24407	2.30	5	1338.77	448.50	14929	2.08
7	1344.50	476.45	155879	2.07	7	1214.75	442.80	33910	2.29
9	1152.46	453.54	153900	2.41	9	1216.02	567.61	40411	2.29

Note: All times are reported in CPU Seconds

Table 6: Parallel Implementation Results for Instance mk5-09

mk5-09									
MinPctGap = 0.1									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Transferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Transferred*	*Speedup*
1	6547.00	-	-	-	1	6547.00	-	-	-
3	4322.51	447.49	33	1.51	3	4158.99	444.77	333	1.57
5	2830.11	882.96	2545	2.31	5	2477.39	611.07	4378	2.64
7	2006.58	887.28	3148	3.26	7	2026.40	639.44	6392	3.23
9	2113.18	1086.79	8477	3.10	9	1786.34	746.29	8680	3.67
MinPctGap = 0.0									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Transferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Transferred*	*Speedup*
1	6547.00	-	-	-	1	6547.00	-	-	-
3	4220.42	606.44	73506	1.55	3	4187.73	543.08	46413	1.56
5	3425.66	1233.84	203551	1.91	5	2365.57	489.95	16604	2.77
7	6804.52	4305.07	693869	0.96	7	1964.53	258.67	194573	3.33
9	2964.82	1159.53	426905	2.21	9	1651.26	298.81	195396	3.96

Note: All times are reported in CPU Seconds

Table 7: Parallel Implementation Results for Instance mk5-10

mk5-10									
MinPctGap = 0.1									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*
1	14213.00	-	-	-	1	14213.00	-	-	-
3	7647.23	308.91	33	1.86	3	8344.24	292.21	333	1.70
5	4608.38	1606.93	2513	3.08	5	4252.45	706.12	7300	3.34
7	3396.25	1289.80	3903	4.18	7	3351.42	931.40	12147	4.24
9	4044.96	1684.24	6870	3.51	9	3170.02	1101.70	13976	4.48
MinPctGap = 0.0									
NodeTransferLim = 100					*NodeTransferLim = 1000*				
Processors	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*	*Processors*	*Total Time*	*Max Wait Time*	*Nodes Trans-ferred*	*Speedup*
1	14213.00	-	-	-	1	14213.00	-	-	-
3	7692.91	307.89	11064	1.85	3	8178.52	509.83	134408	1.74
5	4654.68	447.63	324469	3.05	5	4016.30	233.78	51983	3.54
7	3156.71	278.92	168001	4.50	7	3104.42	227.32	86241	4.58
9	4169.74	621.76	991792	3.41	9	2619.24	187.76	212496	5.43

Note: All times are reported in CPU Seconds

3.5 Concluding Remarks

In this chapter, we reported our computational investigations into the efficacy of hypergraph cutting planes. We chose a suite of test problems that are completely dense and are very difficult to solve by conventional methods. The current implementation of the hypergraph odd hole and clique cuts was not very effective at solving this problem set. We then decided to test the effectiveness of the uniform hyperclique cuts. These performed much better on these dense instances. Our implementation of uniform hypercliques combined with aggregate branching scheme beat CPLEX by a substantial margin. We also experimented with parallel computing. The experiments with parallel implementation yielded deeper insights into how these instances behave.

There are a number of areas that need further research. The question of effective implementation of hypergraph odd hole and clique cuts is still open. We experimented with just one set of problems. More experiments are needed to test how different problem types respond to these cuts. We believe that these cuts along with innovative branching schemes may be effective in solving some of the problems encountered in cancer treatment optimization. More research is also needed for exploiting parallel computers. The world of computing is moving into the direction of increasing parallelism. Clearly, the traditional thinking in optimization needs to be modified to take that into account.

CHAPTER IV

CONCLUSION

4.1 Contributions

In this thesis, we laid down a theoretical framework for obtaining facets of the independent set polytope from conflict hypergraphs. We provided generalized definitions for odd holes, odd antiholes, cliques, webs and antiwebs. Results for maximum vertex packing on hypergraph structures were derived. These yielded valid inequalities, which under certain conditions, were shown to be facet-defining for the entire independent set polytope. Later, the definitions were generalized further to include non-separable structures, and valid inequalities and facet-defining conditions were derived for them. We also derived Chvatal-Gomory ranks for the odd hole and clique inequalities. Finally, we provided a framework for creating separation routines for conflict hypergraphs, and illustrated the concept by extending the odd cycle separation algorithm.

We also reported computational investigations into the efficacy of hypergraph cutting planes. We chose a suite of test problems that are completely dense and are very difficult to solve by conventional methods. The current implementation of the hypergraph odd hole and clique cuts was not very effective at solving this problem set. We then decided to test the effectiveness of the uniform hyperclique cuts. These performed much better on these dense instances. Our implementation of uniform hypercliques combined with aggregate branching scheme beat CPLEX by a substantial margin. We also experimented with parallel computing. The experiments with parallel implementation yielded deeper insights into how these instances behave.

4.2 Future Research

On the theoretical side, many open questions remain. It would be interesting to investigate facet-defining conditions for valid inequalities derived from non-separable odd antiholes, webs and antiwebs. Currently, we have been able to get results only for cliques and odd holes. Another interesting question concerns the Chvatal-Gomory ranks for structures other than cliques and odd holes. It is our conjecture, due in large part to the correspondence between elements of the hypergraph structures and reduced graph, that Chvatal-Gomory ranks for other hypergraph inequalities should be same as those of the corresponding structures. It would also be interesting to investigate the theoretical complexity results for the separation problems for hypergraph structures. In fact, much more research is needed for creating practical separation routines that would be effective in real-world branch-and-cut algorithms.

On the computational side, the question of effective implementation of hypergraph odd hole and clique cuts is still open. There are many ways to implement these cuts. More experiments are needed to study how different problem-sets respond to various branch-and-cut implementation. We believe that these cuts along with innovative branching schemes may be effective in solving some of the problems encountered in cancer treatment optimization. More research is also needed for exploiting parallel computing platforms. It would also be interesting to investigate how the information contained in conflict hypergraphs may be used for designing better branching schemes and heuristics.

REFERENCES

[1] Aardal, K., Bixby, R. E., Hurkens, C. A. J., Lenstra, A. K., and Smeltink, J. W., "Market split and basis reduction: Towards a solution of the cornuejols-dawande instances," in *Integer Programming and Combinatorial Optimization, 7th International IPCO Conference*, pp. 1–16, Springer-Verlag, 1999.

[2] Applegate, D., Bixby, R., Chvátal, V., and Cook, W., "On the solution of traveling salesman problems," *Documenta Mathematica*, vol. Extra Volume Proceedings ICM III, pp. 645–656, 1998.

[3] Ashford, R. W., Connard, P., and Daniel, R. C., "Experiments in solving mixed integer programming problems on a small array of transputers," *Journal of the Operational Research Society*, vol. 43, pp. 519 – 531, 1992.

[4] Atamtürk, A., *Conflict Graphs and Flow Models for Mixed-Integer Linear Optmization Problems.* PhD thesis, Georgia Institute of Technology, Atlanta, U.S.A., 1998.

[5] Atamtürk, A., Nemhauser, G. L., and Savelsbergh, M. W. P., "Conflict graphs in solving integer programming problems," *European Journal of Operational Research*, vol. 121, pp. 40–55, 2000.

[6] Atamtürk, A., Nemhauser, G. L., and Savelsbergh, M. W. P., "The mixed vertex packing problem," *Math. Program.*, vol. 89, pp. 35–53, 2000.

[7] Beasley, J. E., "Or-library: distributing test problem by electronic mail," *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, 1990.

[8] Berge, C., *Graphs and Hypergraphs.* North-Holland, 1973. translated by E. Minieka.

[9] Bixby, R. E., Cook, W., Cox, A., and Lee, E. K., "Parallel mixed integer programming," Tech. Rep. CRPC–TR95554, Rice University, Houston, Texas, 1995.

[10] Bixby, R. E. and Lee, E. K., "Solving a truck dispatching scheduling problem using branch and cut," *Oper. Res.*, vol. 46, no. 3, pp. 355–367, 1994.

[11] Bixby, R., Cook, W., Cox, A., and Lee, E., "Computational experience with parallel mixed integer programming in distributed environment," *Ann. Oper. Res.*, vol. 90, pp. 19–43, 1999.

[12] Borndörfer, R., *Aspects of Set Packing, Partitioning and Covering.* PhD thesis, Technischen Universtät Berlin, Berlin, Germany, 1997.

[13] Cannon, T. L. and Hoffman, K. L., "Large-scale 0-1 linear programming on distributed workstations," *Ann. Oper. Res.*, vol. 22, no. 1-4, pp. 181–217, 1990.

[14] Chu, P. C. and Beasley, J. E., "A genetic algorithm for the multidimensional knapsack problem," *Journal of Heuristics*, vol. 4, pp. 63–86, 1998.

[15] Cornuéjols, G. and Sassano, A., "On the 0,1 facets of the set covering polytope," *Math. Program.*, vol. 43, pp. 45–55, 1989.

[16] Cornuéjols, G. and Dawande, M., "A class of hard small 0-1 programs," *INFORMS Journal on Computing*, vol. 11, no. 2, pp. 284–293, 1999.

[17] Crowder, H., Johnson, E. L., and Padberg, M. W., "Solving large-scale zero-one linear programming problems," *Operations Research*, vol. 31, no. 5, pp. 803–834, 1983.

[18] Easton, T., Hooker, K., and Lee, E. K., "Facets of the independent set polytope," *Math. Program.*, vol. 98, pp. 177–199, 2003.

[19] Eckstein, J., "How much communication does parallel branch and bound need," *INFORMS Journal on Computing*, vol. 9, no. 15 - 29, 1997.

[20] Eckstein, J., "Control strategies for parallel mixed integer branch and bound," in *Supercomputing '94: Proceedings of the 1994 conference on Supercomputing*, (Los Alamitos, CA, USA), pp. 41–48, IEEE Computer Society Press, 1994.

[21] Euler, R., Jünger, M., and Reinelt, G., "Generalizations of cliques, odd cycles and anticycles and their relation to indpendence system polyhedra," *Math. Oper. Res.*, vol. 12, no. 2, pp. 451–462, 1987.

[22] Gendron, B. and Crainic, T. B., "Parallel branch and bound algorithms: survey and synthesis," *Operations Research*, vol. 42, pp. 1042 – 1066, 1994.

[23] Gendron, B. and Crainic, T., "Parallel branch-and-bound algorithm for general integer programming on the cm-5," *SIAM J. Optim.*, vol. 42, pp. 1042–1066, 1994.

[24] Golumbic, M. C., "Interval graphs and related topics," *Disc. Math.*, vol. 55, pp. 113–121, 1985.

[25] Grötschel, M., Lovász, L., and Schrijver, A., *Geometric Algorithms and Combinatorial Optimization.* Berlin: Springer-Verlag, 1993.

[26] Johnson, E. L., Nemhauser, G. L., and Savelsbergh, M. W. P., "Progress in linear programming-based algorithms for integer programming: An exposition," *INFORMS J. Comput.*, vol. 12, no. 1, pp. 2–23, 2000.

[27] Junger, M. and Naddef, D., eds., *Computational Combinatorial Optimization.* Lecture Notes in Computer Science, Berlin: Springer, 2001.

[28] Keleher, P., Dwarkadas, S., Cox, A. L., and Zwaenepoel, W., "Treadmarks: Distributed shared memory on standard workstations and operating systems," in *Proc. of the Winter 1994 USENIX Conference*, pp. 115–131, 1994.

[29] Laurent, M., "A generalization of antiwebs to independence systems and their canonical facets," *Math. Program.*, vol. 45, pp. 97–108, 1989.

[30] Lee, E. K., *Solving a Truck Dispatching Scheduling Problem Using Branch-and-Cut.* PhD thesis, Rice University, Houston, TX, 1993.

[31] Linderoth, J. T., *Topics in Parallel Integer Optimization.* PhD thesis, Georgia Institute of Technology, August 1998.

[32] Marchand, H., Martin, A., Weismantel, R., and Wolsey, L., "Cutting planes in integer and mixed integer programming," *Discrete Appl. Math.*, vol. 123, pp. 397–446, 2002.

[33] Müller, R. and Schulz, A. S., "Transitive packing: A unifying concept in combinatorial optimization," *SIAM J. Optim.*, vol. 13, pp. 335–367, 2002.

[34] Nemhauser, G. L. and Jr., L. T., "Properties of the vertex packing and independence system polyhedra," *Math. Program.*, vol. 6, pp. 48–61, 1974.

[35] Nemhauser, G. L. and Wolsey, L. A., *Integer and Combinatorial Optimization.* New York: John Wiley and Sons, 2001.

[36] Padberg, M. W., "On the facial structure of set packing polyhedra," *Math. Program.*, vol. 5, pp. 199–215, 1973.

[37] Padberg, M. W., "Covering, packing and knapsack problems," *Annals of Discrete Mathematics*, vol. 1, pp. 421–434, 1979.

[38] Papadimitrou, C. H. and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity.* Mineola, New York: Dover Publications, Inc., 1998.

[39] Pruul, E., Nemhauser, G. L., and Rushmeier, R. A., "Branch and bound and parallel computation: a historical note," *Operations Research Letters*, vol. 7, pp. 65 – 69, 1988.

[40] Sekiguchi, Y., "A note on node packing polytopes on hypergraphs," *Oper. Res. Lett.*, vol. 2, pp. 243–247, 1983.

[41] Trotter, L. E., "A class of facet producing graphs for vertex packing polyhedra," *Disc. Math.*, vol. 12, pp. 373–388, 1975.