The Knapsack Problem with Conflict Pair Constraints on Bipartite Graphs and Extensions

Abraham P. Punnen and Jasdeep Dhahan *

Abstract: In this paper, we study the knapsack problem with conflict pair constraints. After a thorough literature survey on the topic, our study focuses on the special case of bipartite conflict graphs. For complete bipartite (multipartite) conflict graphs, the problem is shown to be NP-hard but solvable in pseudo-polynomial time, and it admits an FPTAS. Extensions of these results to more general classes of graphs are also presented. Further, a class of integer programming models for the general knapsack problem with conflict pair constraints is presented, which generalizes and unifies the existing formulations. The strength of the LP relaxations of these formulations is analyzed, and we discuss different ways to tighten them. Experimental comparisons of these models are also presented to assess their relative strengths. This analysis disclosed various strong and weak points of different formulations of the problem and their relationships to different types of problem data. This information can be used in designing special purpose algorithms for KPCC involving a learning component.

Keywords: knapsack problem; stable sets; conflict pair constraints; combinatorial optimization; algorithms

1. Introduction

Let $G = (V, E)$ be an undirected graph, and for each $i \in V = \{1, 2, \ldots, n\}$ an ordered pair $(c_i, a_i)$ is prescribed such that $a_i, c_i \geq 0$ for all $i$. This non-negativity assumption can be relaxed in some of the results discussed in this paper. We refer to $c_i$ as the cost of $i$ and $a_i$ as the weight of $i$. Let $b \in \mathbb{R}^+ \cup \{0\}$ be a given budget. For any $S \subseteq V$, let $c(S) = \sum_{i \in S} c_i$ be its cost and $a(S) = \sum_{i \in S} a_i$ be its weight. Then, the knapsack problem with conflict pair constraints (KPCC) is to find a stable set $S$ in $G$ satisfying $a(S) \leq b$ such that $c(S)$ is as large as possible.

The KPCC can be formulated as a 0-1 programming problem as follows:

Maximize $\mathbf{cx}$

Subject to: $\mathbf{ax} \leq b$

$x_i + x_j \leq 1$ for all $(i, j) \in E$

$x \in \{0, 1\}^n$

where $\mathbf{c} = (c_1, c_2, \ldots, c_n)$, $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{x}^T = (x_1, x_2, \ldots, x_n)$.

The KPCC has been investigated by many researchers, and it is known under different names. Some authors call it the maximum independent set problem with a budget constraint [1–3], while some others call it the disjunctively constrained knapsack problem [4–10]. We use the name knapsack problem with conflict pair constraints since it belongs to the broader class of combinatorial optimization problems with conflict pair constraints [11–20].

Bandyapadhyay [1] notes that the maximum weight stable set problem (MWSS) has many applications in various fields, including scheduling, wireless networks, computer graphics, map labeling, and molecular biology. The budget restriction has a relevant interpretation in most of these applications, and it provides motivation to study KPCC. Moreover, specific applications of KPCC include job scheduling in computers and selecting a non-interfering set of transmitters [1]. Other applications of KPCC also have been
discussed in the literature. Further, KPCC is used in the Dantzig–Wolfe decomposition formulation of the two-dimensional bin packing problem [21].

2. Literature Review

Combinatorial optimization problems with conflict pair constraints have recently received considerable attention from the research community. These include conflict pair versions of the minimum spanning tree problem [11–14,19,20], shortest path problem [14], minimum cost matching problem [14,17], minimum cost bin packing [22,23], and maximum flow problem [18]. KPCC is yet another member in this class of optimization problems. Conflict pair constraints can easily be absorbed into a quadratic objective function. Thus, these types of problems can be formulated as quadratic combinatorial optimization problems. However, the special structure of the conflict pair constraints makes the problem interesting to study independently.

The multiple choice knapsack problem studied by Chandra et al. [24] and Nauss [25] in 1975 is essentially a KPCC, where the underlying graph is a collection of disjoint cliques. The continuous version of this problem was investigated by Ibaraki et al. [26]. Another special case of KPCC, where $c_j = 1$ for all $j$, was mentioned by Bar-Noy et al. [27] in the context of resource allocation. The general problem KPCC was introduced by Yamada, Kataoka, and Watanab [10], and they proposed exact and heuristic algorithms to solve the problem. Hifi and Michrafy [6] presented different exact algorithms based on MIP formulations. Bettinelli, Cacchiani, and Malaguti [28] proposed yet another class of the branch and bound algorithm, along with extensive experimental analysis. Moreover, they presented comparisons of MIP formulations using general purpose solvers. Salem et al. [29] investigated the convex hull of feasible solutions of KPCC, i.e., the KPCC polytope. The authors obtained necessary and sufficient conditions for odd cycle inequalities associated with the stable set problem to be facet-defining for the KPCC polytope. Similar conditions were developed for the cover inequalities of the knapsack polytope in connection with the KPCC polytope. They also analyzed different lifting procedures to obtain additional valid inequalities. Exploiting these inequalities, the authors proposed an effective branch and bound algorithm to solve KPCC and reported the results of extensive computational experiments.

Heuristic algorithms for KPCC have been investigated by many researchers, and the results of extensive experimental analysis with these algorithms are available. Hifi and Michrafy [5] developed a reactive-local-search-based algorithm and also generated a class of test instances based on the ideas from [10]. Akeb et al. [4] proposed local-branching-based heuristics and provided an experimental analysis based on instances generated by [5]. Hifi and Otmani [7] proposed a scatter-search-based metaheuristic algorithm along with results of an experimental analysis using test instances from [5]. Recently, Hifi [9] proposed an iterative-rounding-search-based algorithm. Their experimental results using test instances from [5] were promising and were able to obtain the best known solutions for many benchmark instances. Salem et al. [30] developed a probabilistic tabu search algorithm to solve KPCC and reported extensive computational results.

Pferschy and Schauer [31] investigated KPCC for several special cases of the graph $G$. They showed that KPCC can be solved in pseudo-polynomial time when $G$ is a tree and more generally when $G$ is a graph with bounded treewidth or chordal. Fully polynomial-time approximation schemes (FPTASs) are also derived for these classes of graphs. Further, they proved that KPCC remains strongly NP-hard on perfect graphs. It may be noted that the stable set problem is polynomial-time-solvable on perfect graphs. Bandyapadhyay [1] also considered KPCC on trees, but the algorithm discussed in [31] is faster for this case. Moreover, [1] presents pseudopolynomial algorithms for cycles, forests, $k$-outerplanar graphs, and interval graphs. In another paper, Pferschy and Schauer [32] investigated approximation algorithms for KPCC under various restrictions on $G$. They showed that KPCC admits FPTASs on weakly chordal graphs and is solvable by PTAS (not FPTASs) on planar graphs.
Kalra et al. [2] studied another special case of KPCC, where \( a_j = 1 \) for all \( j = 1, 2, \ldots, n \) and \( G \) is a multipartite graph. They proposed a \( \frac{1}{2} \)-approximation algorithm, where \( k \) is the number of partite sets, and proved that the performance bound established is tight. If \( a_j = 1 \) and \( c_j \) equal to the degree of \( j \), for \( j = 1, 2, \ldots, n \), KPCC reduces to the maximum independent vertex coverage problem (MIVC). The authors proved approximation hardness results for MIVC and showed that MIVC is NP-hard on bipartite graphs. Further, they established bounds on the integrality for the LP relaxation of a clique-based formulation of MIVC. Bandyapadhyay [1] investigated another special case of KPCC, where \( c_j = 1 \) for all \( j \). This problem is called the maximum budget independent set problem (MBIS), and the author presented a factor \( d \) polynomial time approximation algorithm for the problem on \((d+1)\)-claw free graphs.

In KPCC, if we restrict the budget constraint to be satisfied as an equality, we obtain an instance of the exact knapsack problem with conflict pair constraints (KPCC-exact). The feasibility version of this problem was investigated by Milanic and Monnot [3,33], where it was shown that the problem is strongly NP-complete on bipartite graphs with maximum vertex degree of three. The authors also presented a pseudo-polynomial algorithm for the problem in cographs, chordal graphs, and interval graphs.

A generalization of KPCC involving more than one knapsack constraint was studied by Gabriel [34], where a Dantzig–Wolfe decomposition scheme was proposed. Atamtürk et al. [35] used conflict graphs in developing algorithms for 0-1 integer programs.

The quadratic knapsack problem (QKP) [36] can be stated as

Maximize \[ x^T Q x + c^T x \]
Subject to:
\[ a^T x \leq b \]
\[ x \in \{0, 1\}^n \]

where \( Q = (q_{ij}) \) is an \( n \times n \) matrix.

Although the QKP appears to be more general than KPCC, we now observe that QKP and KPCC are essentially equivalent, in the sense that one is a special case of the other.

**Theorem 1.** QKP and KPCC are equivalent.

**Proof.** Define the matrix \( Q = (q_{ij}) \) such that \( q_{ij} = -M \), where \( M \) is a large positive number if \( (i,j) \in E \) and \( q_{ij} = 0 \) otherwise. With this choice of \( Q \), the problem KPCC reduces to QKP. Thus, KPCC is a special case of QKP.

Let us now prove the converse. That is, QKP is a special case of KPCC. This is, however, achieved by appropriate modifications in the well-known reduction of a quadratic unconstrained binary optimization problem (QUBO) to the maximum weight stable set problem [37,38] to handle the budget constraint. Thus, KPCC can be viewed as a generalization of QKP. \( \square \)

For further details on QUBO, refer to [39]. The major contributions of the paper can be summarized as follows.

- We present new complexity results on KPCC along with a thorough state-of-the-art review of the literature. Also, we show that KPCC and QKP are equivalent.
- When the conflict graph is a complete bipartite graph, it is shown that KPCC decomposes into two knapsack problems. As a consequence, we have new polynomially solvable (and pseudo-polynomially solvable) special cases of KPCC and have special cases of KPCC that admit FPTASs. These results are extended to more general classes of graphs that accept a blackbox oracle with special properties. On a star (which is a special bipartite graph), the problem is shown to be NP-hard.
- A new integer programming formulation that unifies and generalizes existing formulations of KPCC on general graphs is given. The strengths of the LP relaxation of this formulation and of special cases are analyzed theoretically.
Different methods are proposed to tighten our general integer programming formulation, and these methods are compared using the general purpose solver Gurobi.

3. KPCC on Bipartite Graphs

Let \( G = (V_1 \cup V_2, E) \) be a bipartite graph with the generic bipartition of its vertex set as \( V_1 \cup V_2 \), where \( V_1 = \{1, 2, \ldots, m\} \) and \( V_2 = \{m + 1, m + 2, \ldots, m + n\} \). Note that KPCC is trivial to solve when \( G \) is a complete graph and is the same as the knapsack problem when \( G \) is a collection of isolated nodes. When \( G \) is a tree, Pferschy and Schauer \cite{31} showed that KPCC is NP-hard but solvable in \( O(nb^2) \) time and \( O(nb) \) space. Further, they showed that the algorithm can be modified to obtain a fully polynomial approximation scheme (FPTASs) for KPCC. Trees, in some sense, can be viewed as an extreme case of sparse connected bipartite graphs. Let us now look at KPCC on the complete bipartite graph \( K_{m,n} = (V_1 \cup V_2, E) \). Although KPCC is trivial on a complete graph, this simplicity does not extend to \( K_{m,n} \).

Theorem 2. KPCC is NP-hard even if the conflict graph is a star (i.e., \( K_{1,n} \)). Further, on a complete bipartite graph, KPCC can be solved in pseudopolynomial time and it admits FPTASs.

Proof. It is not difficult to reduce the knapsack problem to a KPCC on \( K_{1,n} \) and this establishes the first part of the theorem. To prove the second part, note that for any solution \( x \) of a KPCC on \( K_{m,n} \), if \( x_i = 1 \) for any \( i \in V_1 \) then by the conflict pair constraints, \( x_i = 0 \) for all \( i \in V_2 \) and vice versa. Thus, KPCC on \( K_{m,n} \) reduces to two knapsack problems

\[
\text{KP1: Maximize } \sum_{j \in V_1} c_j x_j \\
\text{Subject to: } \sum_{j \in V_1} a_j x_j \leq b \\
x_j \in \{0, 1\} \text{ for all } j \in V_1
\]

and

\[
\text{KP2: Maximize } \sum_{j \in V_2} c_j x_j \\
\text{Subject to: } \sum_{j \in V_2} a_j x_j \leq b \\
x_j \in \{0, 1\} \text{ for all } j \in V_2
\]

The best solution amongst the solutions of KP1 and KP2 solves KPCC on \( K_{m,n} \). Since the knapsack problem can be solved in pseudopolynomial time and it admits FPTASs, the second part of the theorem follows. \( \Box \)

From the above theorem, if KP1 and KP2 are solvable in polynomial time, then the corresponding KPCC on \( K_{m,n} \) is also solvable in polynomial time. If \( a_j = 1 \) for all \( j \in V_1 \cup V_2 \), then obviously KP1 and KP2 are solvable in polynomial time; hence, the corresponding KPCC on \( K_{m,n} \) is also solvable in polynomial time. This is interesting since KPCC on a general bipartite graph is NP-hard even if \( a_j = 1 \) for all \( j \) \cite{31}. Deineko and Woeginger \cite{40} showed that the cross-ratio-ordered bounded knapsack problem on \( n \) variables can be solved in \( O(n) \) time. This, together with Theorem 2, leads to the following theorem.

Theorem 3. If the costs \( c_i \) and weight \( a_i \) for \( i \in V_1 \cup V_2 \) satisfy \( \frac{a_{i+1}}{a_i} \leq \left\lfloor \frac{c_{i+1}}{c_i} \right\rfloor \) for \( i = 1, 2, \ldots, m - 1 \) and \( \frac{a_{i+1}}{a_i} \leq \left\lfloor \frac{c_{i+1}}{c_i} \right\rfloor \) for \( i = m + 1, m + 2, \ldots, m + n - 1 \), then KPCC on \( K_{m,n} \) can be solved in polynomial time.
We can also mix conditions for different solvable cases for KP1 and KP2 to obtain solvable instances of KPCC on $K_{m,n}$. For example, if $a_i = \alpha$ for all $i \in V_1$ and $\frac{a_i + 1}{c_i} \leq \frac{a_i + 1}{c_i} + j$ for $i = m + 1, m + 2, \ldots, m + n - 1$, then KPCC on $K_{m,n}$ can also be solved in polynomial time. Later, we present additional conditions for polynomially solvable special cases of the knapsack problem that can be used in different combinations for KP1 and KP2 to produce polynomially solvable cases of KPCC on $K_{m,n}$.

Let $G = (V_1 \cup V_2, E)$ be an undirected graph such that the subgraphs induced by $V_1$ and $V_2$ belong to a family $\mathcal{F}$ of graphs with specific properties. We call such graphs $\mathcal{F}$-bipartite (see Figure 1 for an example). If an $\mathcal{F}$-bipartite graph contain all edges $(i, j)$ for $i \in V_1, j \in V_2$, then it is called a complete $\mathcal{F}$-bipartite graph. Bonamy et al. [41] studied $\mathcal{F}$-bipartite graphs, where the induced subgraph of $V_1$ is a collection of isolated nodes and the induced subgraph of $V_2$ is a $k$-degenerate graph. Depending on the properties imposed on the family $\mathcal{F}$, the $\mathcal{F}$-bipartite structure can have algorithmic advantages in solving some optimization problems. For example, $\mathcal{F}$ could contain all classes of graphs on which the KPCC can be solved in pseudo-polynomial time. In this case, $\mathcal{F}$ includes graphs that are isolated vertices, trees, complete bipartite graphs, complete graphs, chordal graphs, interval graphs, cographs, and circular arc graphs.

![Figure 1. A complete $\mathcal{F}$-bipartite graph with $V_1 = \{v_1, v_2, v_3, v_4\}$ and $V_2 = \{u_1, u_2, u_3\}$. The subgraph induced by $V_1$ is a cycle and a pendant vertex, and the subgraph induced by $V_2$ is tree (path).](image)

Assume that for a $\mathcal{F}$-bipartite graph, the bipartition $V_1$ and $V_2$ along with the structure of the subgraphs induced by $V_1$ and $V_2$ are given. Then, the results obtained in the previous sections on the solvability of KPCC can be extended to $\mathcal{F}$-bipartite graphs.
Theorem 4. KPCC can be solved in pseudo-polynomial time on a complete $F$-bipartite graph $G = (V_1 \cup V_2, E)$ if KPCC can be solved in pseudo-polynomial time on graphs in $F$. Further, KPCC admits FPTASs on $G$ whenever KPCC admits FPTASs on graphs in $F$.

Proof. The proof of the above theorem is similar to that of Theorem 2, except that instead of KP1 and KP2 we will have problems of the type KPCC but with reduced sizes. However, we present the complete proof of the theorem here to facilitate further related discussions. Let $G_1 = (V_1, E_1)$ be the subgraph of $G$ induced by $V_1$ and $G_2 = (V_2, E_2)$ be the subgraph of $G$ induced by $V_2$. Since every vertex in $V_1$ is connected to every vertex in $V_2$, for any solution $x$ of the KPCC, if $x_i = 1$ for any $i \in V_1$ then $x_i = 0$ for all $i \in V_2$ and vice versa. Thus, KPCC reduces to the following reduced KPCCs.

**KPCC1:** Maximize $\sum_{j \in V_1} c_j x_j$

Subject to:

- $\sum_{j \in V_1} a_j x_j \leq b$
- $x_i + x_j \leq 1$ for $(i, j) \in E_1$
- $x_j \in \{0, 1\}$ for all $j \in V_1$

and

**KPCC2:** Maximize $\sum_{j \in V_2} c_j x_j$

Subject to:

- $\sum_{j \in V_2} a_j x_j \leq b$
- $x_i + x_j \leq 1$ for $(i, j) \in E_2$
- $x_j \in \{0, 1\}$ for all $j \in V_2$

The best solution amongst the solutions of KPCC1 and KPCC2 provides an optimal solution to KPCC. By assuming the properties of $F$, KPCC1 and KPCC2 can be solved in pseudo-polynomial time. The proof of the second part of the theorem also follows in an analogous way.

Theorem 4 is a proper generalization of Theorem 2.

The results discussed above extend in a natural way to complete multi-partite graphs. However, to take any computational advantage we need to know the partite sets a priori or need to compute them. Bipartite graphs can be recognized in polynomial time but recognizing $p$-partite graphs for $p > 2$ is NP-complete [42]. If there are $p$ partite sets, then we will be solving $p$ knapsack problems.

General Bipartite Graphs

We now consider KPCC on a general bipartite graph $G = (V_1 \cup V_2, E)$, where $V_1 = \{1, 2, \ldots, m\}$ and $V_2 = \{m + 1, m + 2, \ldots, m + n\}$. Note that we are considering a maximization problem. Consider the maximization version of a combinatorial optimization problem such that the objective function value of its feasible solutions are positive. Let $x^*$ be any feasible solution to such a problem and $x^0$ be a corresponding optimal solution. Then, $x^*$ is said to be $\epsilon$-optimal if $\frac{OBJ(x^0)}{OBJ(x^*)} \leq \epsilon$, where $OBJ(x)$ is the objective function value of the solution $x$. When $\epsilon = 1$, then $x^*$ is an optimal solution. Consider KP1 and KP2, as indicated in Theorem 2. Let $x_1$ and $x_2$ be $\epsilon$-optimal solutions to KP1 and KP2, respectively. Define the solutions $\hat{x}_1$ and $\hat{x}_2$, where

$$\hat{x}_1 = \begin{cases} x_1 & \text{if } x \in V_1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad \hat{x}_2 = \begin{cases} x_2 & \text{if } x \in V_2 \\ 0 & \text{otherwise} \end{cases}$$
Theorem 5. The best solution amongst $x^1$ and $x^2$ is a $2\epsilon$-optimal solution to KPCC on $G = (V_1 \cup V_2, E)$.

Proof. Let $x^0$ be an optimal solution to KPCC on $G = (V_1 \cup V_2, E)$. Define the solutions $x^*$ and $x^{**}$ such that

$$x^*_j = \begin{cases} x^0_j & \text{if } x \in V_1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad x^{**}_j = \begin{cases} x^0_j & \text{if } x \in V_2 \\ 0 & \text{otherwise} \end{cases}$$

Let $x^{1*}$ be the restriction of $x^*$ to $V_1$ and $x^{2**}$ be the restriction of $x^{**}$ to $V_2$. Then, $x^{1*}$ and $x^{2**}$ are feasible solutions of KP1 and KP2, respectively. Then, by the $\epsilon$-optimality of $\hat{x}_1$ and $\hat{x}_2$ to KP1 and KP2, we have

$$\sum_{j \in V_1} c_j x^1_j \leq \epsilon \left( \sum_{j \in V_1} c_j x^*_j \right) = \epsilon cx^*$$

(1)

and

$$\sum_{j \in V_2} c_j x^2_j \leq \epsilon \left( \sum_{j \in V_2} c_j x^{**}_j \right) = \epsilon cx^{**}$$

(2)

Without a loss of generality, assume $cx^1 \geq cx^2$. Then,

$$cx^0 = cx^* + cx^{**} \leq \epsilon cx^1 + \epsilon cx^2 \leq 2\epsilon cx^1.$$

An approximation algorithm $A$ for a combinatorial optimization problem (in maximization form) is said to be a $\delta$-fully polynomial time approximation scheme ($\delta$-FPTASs) if it guarantees a solution $x^*$ such that $OBJ(x^0) \leq \delta(1 + \epsilon)OBJ(x^*)$, where $x^0$ is an optimal solution and $A$ runs in polynomial time in the input size and $\frac{1}{\epsilon}$ for any $\epsilon > 0$. Thus, a $1 - FPTASs$ is precisely an FPTASs. Since KP1 and KP2 admits FPTASs, from Theorem 5, KPCC admits $2 - FPTASs$. This result is interesting because the problem is strongly NP-complete on bipartite graphs with maximum vertex degree of three $[3,33]$. Further, when KP1 and KP2 are solvable in polynomial time, KPCC on a general bipartite graph can be solved by a polynomial time 2-approximation algorithm. When $a_j = 1$ for all $j$, Theorem 5 reduces to a corresponding result obtained by Kalra et al. [2], and for the special case, they showed that the performance bound is asymptotically tight. We summarize these observations in the following corollary.

Corollary 1. KPCC on a bipartite graph admits 2-FPTASs. Further, when KP1 and KP2 are solvable in polynomial time, then a 2-optimal solution to KPCC on a bipartite graph can be obtained in polynomial time, and in this case the bound 2 is asymptotically tight.

The concept of $F$-bipartite graphs can be extended to the more general case of $F$-multipartite graphs. Let $G = \left( \bigcup_{i=1}^p V_i, E \right)$ be an undirected graph such that the subgraphs induced by $V_i$ for any $k = 1, 2, \ldots, p$ belong to the family $F$. We call $G$ a $F$-multipartite graph. If $(i,j) \in E$ for any $i \in V_r, j \in V_s$ for $r \neq s, r,s = 1,2,\ldots,p$, then $G$ is called a complete $F$-multipartite graph. Theorem 4 and the discussions that follow extend in a natural way to complete $F$-multipartite graphs.

4. Integer Programming Formulations

Let us now discuss some integer programming formulations of the KPCC. For each vertex $i \in V$, let $A(i) = \{ j \in V : (i,j) \in E \}$ and the sets $N_i(1), N_i(2), \ldots, N_i(p_i)$ be a given partition
of \( A(i) \), where \( N_i(k) \neq \emptyset \) for any \( k = 1, 2, \ldots, p_i \), \( N_i(r) \cap N_i(s) = \emptyset \) whenever \( r \neq s \), and \( A(i) = \bigcup_{k=1}^{p_i} N_i(k) \). Then, KPCC can be formulated as the 0-1 programming problem

\[
\text{StarIP: Maximize } \sum_{j \in V} c_j x_j \\
\text{Subject to: } \sum_{j \in V} a_j x_j \leq b \\
\quad \sum_{j \in N_i(k)} x_j \leq |N_i(k)|(1 - x_i) \text{ for } k = 1, 2, \ldots, p_i, i = 1, 2, \ldots, n \\
x_j \in \{0, 1\} \text{ for all } j \in V
\]

The above formulation reduces to the standard integer programming formulation of the KPCC when each \( N_i(k) \) is singleton, i.e., \( p_i = |A(i)| \) for all \( i \). When \( p_i = 1 \) for all \( i \) then \( N_i(1) = A(i) \), and the above integer program reduces to

\[
\text{MaxStarIP: Maximize } \sum_{j \in V} c_j x_j \\
\text{Subject to: } \sum_{j \in V} a_j x_j \leq b \\
\quad \sum_{j \in A(i)} x_j \leq |A(i)|(1 - x_i) \text{ for } i = 1, 2, \ldots, n \\
x_j \in \{0, 1\} \text{ for all } j \in V
\]

Thus, our general integer programming framework StarIP provides a class of integer programming formulations of KPCC that subsumes two well-studied formulations.

**Theorem 6.** Among all of the choices of the partition \( N_i(1), N_i(2), \ldots, N_i(p_i) \) for StarIP, the standard formulation is the strongest and MaxStarIP is the weakest in terms of their LP relaxation values.

**Proof.** Let \( x^* = (x_1^*, x_2^*, \ldots, x_n^*) \) be any feasible solution to the LP relaxation of the standard formulation. Then,

\[
x_j^* + x_j^* \leq 1 \text{ for all } (i, j) \in E
\]

We successfully show that \( x^* \) is a feasible solution to the LP relaxation of StarIP for any partition \( N_i(1), N_i(2), \ldots, N_i(p_i) \). Choose a set \( N_i(k) \) for the vertex \( i \). Now, by adding the edge inequalities from (5) for all \( (i, j) \in E \) such that \( j \in N_i(k) \), we obtain

\[
\sum_{j \in N_i(k)} (x_j^* + x_j^*) \leq 1 = \sum_{j \in N_i(k)} x_j^* \leq |N_i(k)|(1 - x_i^*).
\]

Thus, any feasible solution to the standard integer programming formulation of the KPCC is also a solution to the StarIP. Thus, the LP relaxation of the standard formulation is the strongest among the LP relaxation of any formulation within the class of StarIP. The second part of the theorem can be proved in an analogous way. \( \square \)

Note that, for the LP relaxation of StarIP (for any choice of the partition of \( A(i) \)), \( x_i = \frac{1}{2} \) for all \( i \) as a feasible solution, provided that \( \frac{1}{2} \sum_{j \in V} a_j \leq b \). One way to strengthen or tighten StarIP is by computing upper bounds on the stability number of the subgraph \( G(i, k) \) induced by \( N_i(k) \). For example, if this subgraph is perfect, we can calculate the stability number exactly. If \( a \) is the largest cardinality of a matching in this graph, then \( |N_i(k)| \) can be replaced by \( |N_i(k)| - a \). Another way to tighten the formulation StarIP is as follows: let \( \gamma(i, k) \) be the objective function value of the LP relaxation of the standard formulation of KPCC restricted to \( G(i, k) \), where \( c_i = 1 \) for all \( i \). Then, \( |N_i(k)| \) can be replaced by \( |\gamma(i, k)| \). This number can be viewed as a fractional budgeted stability number.

We can use other similar upper bounds to strengthen StarIP. For a review of different upper
bounds that can be easily calculated, we refer to [43]. By tightening StarIP this way, we can obtain a better formulation for KPCC.

Let \( A'(i) = \{ j \in V : (i, j) \in E, j < i \} \) and \( N_i(1), N_i(2), \ldots, N_i(p_1) \) be a given partition of \( A'(i) \). We can replace \( N_i(k) \) by \( N'_i(k) \) in StarIP to obtain a valid formulation, say StarIP', of KPCC. Likewise, we can replace \( A(i) \) by \( A'(i) \) in MaxStarIP to obtain a valid formulation, say MaxStarIP', for KPCC. The formulation MaxStarIP' has been studied by Hifi et al. [6]. The coefficient matrices of these modified formulations are sparser. However, tightening the resulting formulations by using upper bounds as discussed above are likely to yield weaker LP relaxations, in general, compared to tightening the corresponding formulation of StarIP and MaxStarIP. Moreover, any possible computational advantage achieved by sparsity over the suspected weaker LP relaxation needs to be analyzed experimentally.

Based on the ideas used in Theorem 2, we now present a single integer programming formulation of KPCC on \( K_{m,n} \).

\[
\text{KPCC}_{m,n}: \text{Maximize } \sum_{j \in V_1 \cup V_2} c_j x_j \\
\text{Subject to: } \sum_{j \in V_1} a_j x_j \leq b \\
\sum_{j \in V_2} a_j x_j \leq b \\
\sum_{j \in V_1} x_j \leq m(1 - x_i) \text{ for } i \in V_2 \\
\sum_{j \in V_2} x_j \leq n(1 - x_i) \text{ for } i \in V_1 \\
x_j \in \{0, 1\} \text{ for all } j \in V_1 \cup V_2
\]

Note that constraint (6) guarantees that if any \( x_i = 1 \) for \( i \in V_2 \) then \( x_i = 0 \) for all \( i \in V_1 \). Likewise, constraint (7) guarantees that if any \( x_i = 1 \) for \( i \in V_1 \) then \( x_i = 0 \) for all \( i \in V_2 \).

4.1. Experimental Analysis

The formulation MaxStarIP' has been studied and used in algorithms to solve KPCC [6]. There are also effective branch-and-bound algorithms to solve KPCC for general graphs [6,28,44]. However, to the best of our knowledge, there is no computational study available that compares the different formulations of KPCC. Note that the complexity of KPCC can vary significantly depending on the structure of the underlying graph and the level of tightness of the budget constraint. Also, tight sufficiently computable upper bound estimates on the stability number of a graph can affect the efficacy of our formulations StarIP and MaxStarIP. The following Table 1 provides some easily computable upper bounds on the stability number of the graph \( G = (V, E) \).

In the table above, \( \Delta \) denotes the maximum degree and \( \delta \) denotes the minimum degree of vertices in the graph \( G = (V, E) \).

Recall that the integer programming formulation MaxStarIP belongs to the class StarIP. Therefore, it can be strengthened by replacing \( |A(i)| \) with an upper bound on the stability number of the subgraph \( G(i) \) of \( G \) induced by \( A(i) \). When MaxStarIP is strengthened using the bound UB\( k \) obtained from \( G(i) \) to replace \( |A(i)|, k = 1, 2, 3, 4, 5, 6 \), we denote the resulting MaxStarIP formulation by MaxStarIP\( k \). Thus, in addition to MaxStarIP, we have the strengthened formulations MaxStarIP1, MaxStarIP2, MaxStarIP3, MaxStarIP4, MaxStarIP5, and MaxStarIP6. The major goals of the experimental analysis were to:

1. Compare the relative performance of the standard KPCC formulation given in the introduction, MaxStarIP, and its variations MaxStarIP\( k \), \( k = 1, 2, 3, 4, 5, 6 \) using the general purpose solver GUROBI.
2. Study the impact of the tightness of the budget constraints on the formulations that we compare.
3. Study the impact of sparsity of \( G \) on the formulations that we compare.
4. Explore the heuristic value of the formulations.

It may be noted that UB1 will be at least as tight as UB2 and that UB4 applies to a connected graph. For these reasons, we will not use UB2 and UB4 in our experiments.

Table 1. Easily computable stability number bounds.

<table>
<thead>
<tr>
<th>Bound</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB1</td>
<td>$\lfloor \gamma(G) \rfloor$</td>
</tr>
<tr>
<td>UB2</td>
<td>$\lfloor \alpha_f(G) \rfloor$</td>
</tr>
<tr>
<td>UB3</td>
<td>$\frac{1}{2} + \sqrt{\frac{1}{4} + \frac{1}{2} + \frac{</td>
</tr>
<tr>
<td>UB4</td>
<td>$\frac{(\Delta - 1)n + 1}{\Delta}$</td>
</tr>
<tr>
<td>UB5</td>
<td>$</td>
</tr>
<tr>
<td>UB6</td>
<td>$</td>
</tr>
</tbody>
</table>

All of the computational experiments were carried out on a PC with the Windows 10 Enterprise 64-bit operating system, Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, and 16.0 GB of memory. The calling program was written in Python (Python 3.6.1) and called GUROBI 9.0 using the interface gurobipy. The time limit for GUROBI was set to 3600 s. Most of the test problems were solved to optimality within this time limit, except some large or structured instances.

4.2. Benchmark Instances

Our experiments used benchmark instances selected from Bettinelli et al. [28] and instances generated using the DIMACS clique graph library [48]. The instances of Bettinelli et al. [28] have three data sets: R1, R2, and R10. Each data set consists of eight classes, and each class has many instances. For the data set R1, in the first four classes, the items have a uniformly distributed weight from the range [20, 100], and the knapsack capacity is set to $b = 150$. The number $n$ of items is 120, 250, 500, and 1000, respectively. The last four classes have uniformly distributed weights from [250, 500], and the knapsack capacity is set to $b = 1000$. The conflict graphs were generated randomly. For the data set R1, classes 1 to 4 have a total of 360 instances and classes 5 to 8 have a total of 360 instances. Thus, the data set R1 has a total of 720 instances. The data set R3 is similar to R1 except that the budget $b$ is set to $3 \times 150 = 450$. Likewise, the data set R10 is almost identical to R1 except that the budget $b$ is set to $10 \times 150 = 1500$. For details on these instances, we refer to the paper [28].

For our computational experiments, we selected instances from the first four classes of each of the data sets R1, R3, and R10. We categorized these instances further into small, medium, and large. The small instances consist of conflict graphs with 120 to 250 vertices, the medium instances consist of conflict graphs with 500 vertices, and the large instances consist of conflict graphs with 1000 vertices. From each category, we selected 120 instances using a prescribed selection rule. We refer to the aforementioned class of instances as random data instances.

The DIMACS clique graph library consists of unweighted graphs. We generated two sets of KPCC benchmarks out of these instances, called positively correlated problems and negatively correlated problems. For positively correlated problems, the costs and weights have the same ordering, whereas for the negatively correlated problems, the costs and weights have opposite ordering. We used graphs that consist of 171 to 1024 vertices. These instances are further classified into small, medium, and large sizes. The small instances consist of graphs with 171 to 378 vertices, the medium instances consist of graphs with 400 to 500 vertices, and the large instances consist of 700 to 1024 vertices. There are 23 small instances, 24 medium instances, and 13 large instances. The DIMACS graphs were originally generated as clique benchmarks. To obtain stable set instances, we took the complement of these graphs. Let $G = (V, E)$ be such a complement graph. For each $i \in V$, we generated the cost $c_i$ and the weight $a_i$ as uniformly distributed random integers in the range $[1, |V|]$. To obtain positively correlated data, we first sort the vectors $c$ and $a$ in non-decreasing order.
and then apply the same random index permutation to both vectors. Negatively correlated instances are constructed in a similar manner but with opposite ordering. The budgets for the correlated instances are set as follows. First, we generate a random subset $S$ of the vertices of $G$ such that the cardinality of $S$ is the known size of the maximum cardinality stable set on $G$ and set $b = \sum_{i \in S} a_i$. Then, instances are generated by setting the budget equal to the value $b$ scaled by the factors 0.25, 0.50, and 0.75. The instances we used in our experiments are available at https://github.com/jasdeepdhahan/kpcc_instances.git.

4.3. Experimental Results

Let us now discuss the experimental outcomes of our analysis. First, we compared the standard KPCC formulation given in the introduction to MaxStarIP and its variations MaxStarIP1, MaxStarIP3, MaxStarIP5, and MaxStarIP6 using all of the test instances. Table 2 provides some insight into the relative strength of our formulations based on average running times (in seconds). The numbers marked in bold letters correspond to the lowest computational time taken. The experimental results disclose that there is no consistently superior formulation of KPCC among the ones we tested.

Table 2. Average running times (in seconds).

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Standard</th>
<th>MaxStarIP</th>
<th>MaxStarIP1</th>
<th>MaxStarIP3</th>
<th>MaxStarIP5</th>
<th>MaxStarIP6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Medium</td>
<td>547.0358</td>
<td>546.1085</td>
<td>532.7913</td>
<td>542.4379</td>
<td>562.4661</td>
<td>540.5295</td>
</tr>
<tr>
<td>Random Large</td>
<td>1165.9380</td>
<td>1232.1670</td>
<td>1116.0310</td>
<td>1267.2090</td>
<td>1235.8030</td>
<td>1284.1800</td>
</tr>
<tr>
<td>Positive Small</td>
<td>0.4756</td>
<td>0.4833</td>
<td>0.4683</td>
<td>0.4874</td>
<td>0.4883</td>
<td>0.4606</td>
</tr>
<tr>
<td>Positive Medium</td>
<td>117.7284</td>
<td>108.5378</td>
<td>112.0652</td>
<td>105.2428</td>
<td>105.8007</td>
<td>109.7333</td>
</tr>
<tr>
<td>Positive Large</td>
<td>499.7824</td>
<td>506.0453</td>
<td>417.1473</td>
<td>418.3175</td>
<td>446.0069</td>
<td>407.9784</td>
</tr>
<tr>
<td>Negative Small</td>
<td>1.3642</td>
<td>1.9495</td>
<td>1.7018</td>
<td>1.9759</td>
<td>1.7312</td>
<td>1.6509</td>
</tr>
<tr>
<td>Negative Medium</td>
<td>427.9115</td>
<td>499.4719</td>
<td>443.6538</td>
<td>431.8717</td>
<td>432.7311</td>
<td>422.1719</td>
</tr>
<tr>
<td>Negative Large</td>
<td>1866.9380</td>
<td>1937.1970</td>
<td>1780.2640</td>
<td>1723.5700</td>
<td>1698.0620</td>
<td>1742.0950</td>
</tr>
</tbody>
</table>

The efficacy of various formulations could also be dependent on how tight the budget constraint is and what special structure the underlying graph has. Table 3 presents the experimental results after classifying the budget constraint in the test instances as tight, very tight, and moderately tight. Table 4 presents the experimental results after classifying the random test instances based on tightness of the budget constraints as well as the density of the underlying graph. We sometimes refer to a KPCC instance as sparse or dense. A sparse (dense) KPCC instance has a sparse (dense) underlying graph. The results on the correlated test instances are classified only with respect to budget tightness since the density of the underlying graphs was uniform and generally sparse. These results are presented in Table 5.

After this categorized analysis, the standard formulation appears as the preferred formulation overall, if one is constrained to recommend only one formulation. However, it may be noted that MaxStarIP appears to be more suited for sparse small instances with the tightest and loosest budgets. In addition, MaxStarIP outperforms the standard formulation on sparse medium and large instances with the tightest budgets. For our medium and large instances, as the underlying graph density increased the standard formulation became more robust than MaxStarIP. It is also important to point out that 8 out of 120 medium-sized instances were not solved to optimality within the time limit provided, either by the standard formulation or by the MaxStarIP type formulations. In addition, 30 of the 120 large instances with the loosest budget were not solved to optimality within the time limit specified by either of the aforementioned formulations.

Now, let us compare the standard formulation and MaxStarIP type formulations over the correlated data. We found that the positively correlated instances were easier to solve than the negatively correlated instances. Increasing the budget within limits generally increased the running time of our formulations. Large negatively correlated instances often hit the time limit set for the experiments. The standard formulation generally outperformed
MaxStarIP over our negatively correlated data set. We further observed that MaxStarIP is slightly better than the standard formulation for negatively correlated large instances with the loosest budget. For positively correlated data, MaxStarIP is best suited for medium instances and the standard formulation is the strongest for our small and large instances.

It may also be noted that the standard formulation is the most robust in terms of running times, followed by MaxStarIP1, and there is no clear formulation that wins third place. MaxStarIP1 outperformed our other KPCC formulations over denser and larger random instances, while the standard formulation showed superiority on sparser large random instances. Each of the formulations MaxStarIP, MaxStarIP3, MaxStarIP5, and MaxStarIP6 were found to be the preferred formulation for at least one of the random data set classifications in Table 4. Based on this, we conclude that there is no one formulation that is best suited for our positively correlated data set. The standard formulation is best suited for our medium and small negatively correlated data. In addition, MaxStarIP3 outperformed the other formulations over our larger negatively correlated instances with the tightest budgets, and MaxStarIP5 outperformed the other formulations in Table 5 for the remaining large negatively correlated KPCC instances.

Table 3. Average running times (in seconds) by budget.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Budget</th>
<th>Standard</th>
<th>MaxStarIP</th>
<th>MaxStarIP1</th>
<th>MaxStarIP3</th>
<th>MaxStarIP5</th>
<th>MaxStarIP6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Small</td>
<td>very tight</td>
<td>0.17025</td>
<td>0.2180</td>
<td>0.1535</td>
<td>14.48525</td>
<td>14.46275</td>
<td>14.4683</td>
</tr>
<tr>
<td>Random Small</td>
<td>tight</td>
<td>0.8165</td>
<td>1.4390</td>
<td>0.98075</td>
<td>0.93225</td>
<td>0.95875</td>
<td>1.0078</td>
</tr>
<tr>
<td>Random Medium</td>
<td>very tight</td>
<td>1.2145</td>
<td>1.78725</td>
<td>1.602</td>
<td>1.89275</td>
<td>1.769</td>
<td>1.87975</td>
</tr>
<tr>
<td>Random Medium</td>
<td>tight</td>
<td>24.9155</td>
<td>47.09275</td>
<td><strong>21.021</strong></td>
<td>51.623</td>
<td>63.196</td>
<td>37.62275</td>
</tr>
<tr>
<td>Random Medium</td>
<td>moderate tight</td>
<td>1614.97725</td>
<td>1589.4455</td>
<td>1575.751</td>
<td>1573.798</td>
<td>1622.4325</td>
<td><strong>1582.0860</strong></td>
</tr>
<tr>
<td>Random Large</td>
<td>very tight</td>
<td>5.5705</td>
<td>10.0915</td>
<td>12.59575</td>
<td>10.38025</td>
<td>10.95975</td>
<td>11.4418</td>
</tr>
<tr>
<td>Random Large</td>
<td>tight</td>
<td>491.50875</td>
<td>593.444</td>
<td>484.346</td>
<td>806.51925</td>
<td><strong>796.03675</strong></td>
<td>837.51325</td>
</tr>
<tr>
<td>Random Large</td>
<td>moderate tight</td>
<td>3000.735</td>
<td>3092.9655</td>
<td><strong>2851.1515</strong></td>
<td>2964.729</td>
<td>2900.411</td>
<td>3003.5857</td>
</tr>
<tr>
<td>Positive Small</td>
<td>very tight</td>
<td>0.2232</td>
<td>0.1991</td>
<td>0.2173</td>
<td>0.2155</td>
<td>0.2045</td>
<td>0.2245</td>
</tr>
<tr>
<td>Positive Small</td>
<td>tight</td>
<td>0.4123</td>
<td><strong>0.3786</strong></td>
<td>0.4532</td>
<td>0.4936</td>
<td>0.4955</td>
<td>0.4745</td>
</tr>
<tr>
<td>Positive Small</td>
<td>moderate tight</td>
<td>0.7914</td>
<td>0.8723</td>
<td>0.7345</td>
<td>0.7532</td>
<td>0.7650</td>
<td><strong>0.6827</strong></td>
</tr>
<tr>
<td>Positive Medium</td>
<td>very tight</td>
<td>5.4561</td>
<td>4.7843</td>
<td>3.5337</td>
<td>3.6765</td>
<td>3.6857</td>
<td>3.7887</td>
</tr>
<tr>
<td>Positive Medium</td>
<td>tight</td>
<td>47.92785</td>
<td>33.6848</td>
<td>35.3148</td>
<td>32.3804</td>
<td><strong>29.4691</strong></td>
<td>32.3991</td>
</tr>
<tr>
<td>Positive Medium</td>
<td>moderate tight</td>
<td>299.8013</td>
<td>287.1443</td>
<td>297.34522</td>
<td><strong>279.6713</strong></td>
<td>284.2474</td>
<td>293.0122</td>
</tr>
<tr>
<td>Positive Large</td>
<td>very tight</td>
<td><strong>29.4941</strong></td>
<td>35.5012</td>
<td>41.20824</td>
<td>45.17471</td>
<td>47.0088</td>
<td>41.72882</td>
</tr>
<tr>
<td>Positive Large</td>
<td>moderate tight</td>
<td>1093.12675</td>
<td>1082.2853</td>
<td>918.3300</td>
<td>919.5976</td>
<td>959.8629</td>
<td><strong>908.9182</strong></td>
</tr>
<tr>
<td>Positive Large</td>
<td>tight</td>
<td>376.7265</td>
<td>400.34941</td>
<td>291.9035</td>
<td>290.1800</td>
<td>331.1488</td>
<td>273.2882</td>
</tr>
<tr>
<td>Negative Small</td>
<td>very tight</td>
<td>0.3973</td>
<td>0.4814</td>
<td>0.52410</td>
<td>0.5064</td>
<td>0.4745</td>
<td>0.4614</td>
</tr>
<tr>
<td>Negative Small</td>
<td>tight</td>
<td><strong>0.9173</strong></td>
<td>1.2236</td>
<td>1.17409</td>
<td>1.2859</td>
<td>1.13911</td>
<td>1.3295</td>
</tr>
<tr>
<td>Negative Small</td>
<td>moderate tight</td>
<td><strong>2.7782</strong></td>
<td>4.1436</td>
<td>3.4073</td>
<td>4.1354</td>
<td>3.5800</td>
<td>3.1618</td>
</tr>
<tr>
<td>Negative Medium</td>
<td>tight</td>
<td><strong>266.1083</strong></td>
<td>292.6083</td>
<td>276.6429</td>
<td>274.4096</td>
<td>299.2258</td>
<td>270.0600</td>
</tr>
<tr>
<td>Negative Medium</td>
<td>moderate tight</td>
<td>1011.1958</td>
<td>1195.2896</td>
<td>1046.4300</td>
<td>1011.9479</td>
<td>989.7604</td>
<td><strong>987.5854</strong></td>
</tr>
<tr>
<td>Negative Large</td>
<td>very tight</td>
<td>721.0029</td>
<td>857.0400</td>
<td>667.3035</td>
<td><strong>649.35523</strong></td>
<td>666.8012</td>
<td>658.71412</td>
</tr>
<tr>
<td>Negative Large</td>
<td>tight</td>
<td>2095.2776</td>
<td>2210.1188</td>
<td>2010.5912</td>
<td>1965.06882</td>
<td><strong>1879.2935</strong></td>
<td>1953.7835</td>
</tr>
<tr>
<td>Negative Large</td>
<td>moderate tight</td>
<td>2784.5333</td>
<td>2744.4318</td>
<td>2662.8976</td>
<td>2556.2859</td>
<td><strong>2548.0912</strong></td>
<td>2611.7882</td>
</tr>
</tbody>
</table>
The frequency distribution of the fastest running times can provide further insight into the complexity of KPCC that averaging out the running times could not provide. For this analysis, we consider only the test instances that were solved to optimality within the time limit specified. In the event of a tie, any formulations in a tie for the fastest running time were considered to be equally good. From Table 6, we observe that the standard formulation is the preferred formulation followed by MaxStarIP1, and there is no clear winner for third place, based on frequency analysis.
Table 6. Frequency distribution of fastest running times.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Standard</th>
<th>MaxStarIP</th>
<th>MaxStarIP1</th>
<th>MaxStarIP3</th>
<th>MaxStarIP5</th>
<th>MaxStarIP6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Small</td>
<td>62</td>
<td>12</td>
<td>38</td>
<td>17</td>
<td>25</td>
<td>14</td>
</tr>
<tr>
<td>Random Medium</td>
<td>41</td>
<td>14</td>
<td>29</td>
<td>9</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>Random Large</td>
<td>41</td>
<td>6</td>
<td>21</td>
<td>6</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>Positive Small</td>
<td>22</td>
<td>2</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Positive Medium</td>
<td>16</td>
<td>15</td>
<td>9</td>
<td>8</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Positive Large</td>
<td>20</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>Negative Small</td>
<td>35</td>
<td>5</td>
<td>10</td>
<td>3</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Negative Medium</td>
<td>23</td>
<td>4</td>
<td>16</td>
<td>6</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Negative Large</td>
<td>15</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

5. Conclusions

In this paper, we studied the KPCC from various points of views. After a thorough literature survey on the topic, we focussed our attention on bipartite conflict graphs. For a complete bipartite (multipartite) graphs, the problem is shown to be NP-hard but solvable in pseudo-polynomial time and admits FPTASs. Extensions of these results to more general classes of graphs are also presented. We then introduced an integer programming model for KPCC with general conflict graphs, which generalizes and unifies existing formulations. The strength of the LP relaxations of these formulations are analyzed and also discussed in different ways to tighten them. An experimental comparison of our models is also presented to assess their relative strengths. This analysis disclosed various strong and weak points of our formulations and their linkages to the structure of the problem data. The analysis shows some preference towards the standard formulation; none of the formulations uniformly dominated others in all categories of the test data. In any case, the insights generated could be used effectively in the design of special purpose algorithms for KPCC involving some learning components. Such approaches proved useful in the case of other combinatorial optimization problems (e.g., [49]). This study can be followed on with future work.

Author Contributions: Conceptualization, A.P.; methodology, A.P. and J.D.; software, J.D.; validation, J.D.; formal analysis, A.P. and J.D.; investigation, A.P. and J.D.; resources, A.P.; data curation, J.D.; writing – original draft preparation, A.P. and J.D.; writing—review and editing, A.P.; visualization, J.D.; supervision, A.P.; project administration, A.P.; funding acquisition, A.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by an NSERC discovery grant awarded to Abraham P. Punnen.

Data Availability Statement: Our data is available to the public from “https://github.com/jasdeepdhahan/kpcc_instances.git”.

Conflicts of Interest: The authors declare no conflict of interest.

References


25. Nauss, R.M. The 0-1 Knapsack Problem with Multiple Choice Constraints; University of Missouri-St. Louis: St. Louis, MO, USA, 1975; (Revised in 1976).


34. Gabrel, V. Dantzig-Wolfe Decomposition for Linearly Constrained Stable Set Problem; hal-00116732; France 2006. Available online: https://hal.science/hal-00116732/ (accessed on 23 February 2024).


43. Willis, W. Bounds for the Independence Number of a Graph. Master’s Thesis, College of Humanities and Sciences, Virginia Commonwealth University, Richmond, VA, USA, 2011.


**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.