GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE FOR THE MAXIMUM CO-K-PLEX PROBLEM

By

AMOL ATMARAM BHAVE

Bachelor of Engineering

Shivaji University

Kolhapur, India

2006

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2010

GREEDY RANDOMIZED ADAPTIVE SEARCH

PROCEDURE FOR

THE MAXIMUM CO-K-PLEX PROBLEM

Thesis Approved:

Dr. Balabhaskar Balasundaram

Thesis Adviser

Dr. Tieming Liu

Dr. Ricki Ingalls

Dr. Mark E. Payton

Dean of the Graduate College

.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance and the help of several individuals who contributed and extended their valuable assistance in the completion of this research study.

First and foremost, my utmost gratitude and most sincere thanks to my graduate advisor and mentor, Dr. Balabhaskar (Baski) Balasundaram for his constant support, enthusiasm and patience he has provided throughout my masters and research studies at Oklahoma State University. His individuality, acquaintance and passion in area of operations research not only provided an ideal atmosphere for learning and research but also influenced me in shaping my professional life. I appreciate all his endless contributions to make my masters experience a memorable one and all my admiration goes to him.

I owe my deepest gratitude to Dr. Tieming Liu and Dr. Ricki Ingalls for serving on my graduate committee and for their support and patience during my research studies. Particularly, I am deeply indebted to them for making my thesis proposal defense experience productive by putting great efforts and giving valuable suggestions. I would also like to thank all the faculty in the School of Industrial Engineering and Management, for making my masters studies at Oklahoma State University a great experience.

I would like to thank Ameya Dhaygude and Sameer Mangalvedhe for their constant support and for being wonderful colleagues at Oklahoma State University. I would like to thank Harshal Patwardhan, Ganesh Walunj, Vivekanand Sharma, Sharad Yadav, Sushant Goltkar, Niketan Mahadik, Abhishek Dhuri, Alok Dange, Kunal Divekar, Ameya Phadake and Kedar

Vilankar for being wonderful friends and source of inspiration. Also I would like to thank everybody for their involvement in the successful realization of my thesis, as well as expressing my apology that I could not mention everyone personally.

Last but not the least, I would like to express my gratitude to my family, without their endless support and love it could not have been achieved. I am ever grateful and indebted to my parents for always giving me more than I wanted, more than I deserved and more than they could. Finally, I would like to thank God for making me part of such a wonderful family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A graph $G = (V, E)$ is defined by a set of vertices (V), and a set of edges (E) between pairs of vertices. Graphs provide a convenient and simple way to represent massive data sets arising from complex and large-scale real world systems by allowing association of attributes with its vertices and edges. Properties of graphs provide useful information about the internal structure of the system by identifying meaningful patterns in the dataset. It also facilitates these complex and large scale problems to be solved using combinatorial optimization approaches. Some examples of networks are as follows, electrical and power networks, wireless networks, logistical networks, transportation networks, rail and airline service networks, telecommunication networks, computer networks, biological networks and many more.

A sociogram (visualization of a social network by using a graph) was first presented by Moreno in [1] to analyze structural properties and patterns of group interactions within the network. The term social network was first coined by Barnes in [2]. “A social network is usually represented by a graph, in which the set of vertices corresponds to the “actors” in a social network and the edges correspond to the “ties” between them” [3]. Actors can be people or groups of people, organizations and examples of tie between people or groups of people can be a relationship and between organizations can be various transactions between them.

Social Network Analysis (SNA) is a tool in modern sociology and studies social behavior of the actors in the social network. Thus, social networks can be easily and conveniently modeled as a graph to study and analyze the behavior of social systems. Social network analysis aims at understanding relationships within groups and analyzes useful patterns in the social network by visualization and modeling. For example, insurance companies are interested to find group of people who are most likely to buy their insurance policy by customizing their products on basis of individual attributes such as age, occupation, gender etc. Social network analysis basically relies on identifying *cohesive subgroups* in a social network depending upon the problem definition. Notion of cohesive subgroups is important in SNA from sociological and data mining perspective due to their desirable properties as follows (i) *familiarity* in subgroup members (few strangers) (ii) *reachability* in subgroup members (quick communication) and (iii) *robustness* in subgroup (not easily destroyed by removing members). We can model these properties using graph theoretic concepts of vertex degrees, pair-wise distances/diameters and vertex connectivity, respectively. Figure 1 generated using the "*social graph*" application of *Facebook*, illustrates the notion of cohesive subgroups. In Figure 1, each node represents a friend of mine and there exists an edge between the nodes if they know each other. It is straightforward to identify three cohesive subgroups in Figure 1. The largest cohesive subgroup represents my group of friends at Oklahoma State University. The middle and bottom subgroups represent my group of friends from my hometown and bachelors college respectively.

Let us consider the network-based model of stock market [4, 5], which was built from pair-wise correlations between all stocks traded in U.S. stock market over a specified period of time. The main aim behind the analysis of the market graph is to find diversified portfolios that consistently outperform the market trends. The stocks in the U.S. stock market are represented by vertices in the market graph and an edge between two vertices indicates that the two stocks are correlated above a user-defined correlation threshold value θ, $-1 < \theta < 1$. It is shown that edge density in a constructed market graph is inversely proportional to θ, i.e., edges in the graph decreases as θ increases in [6].

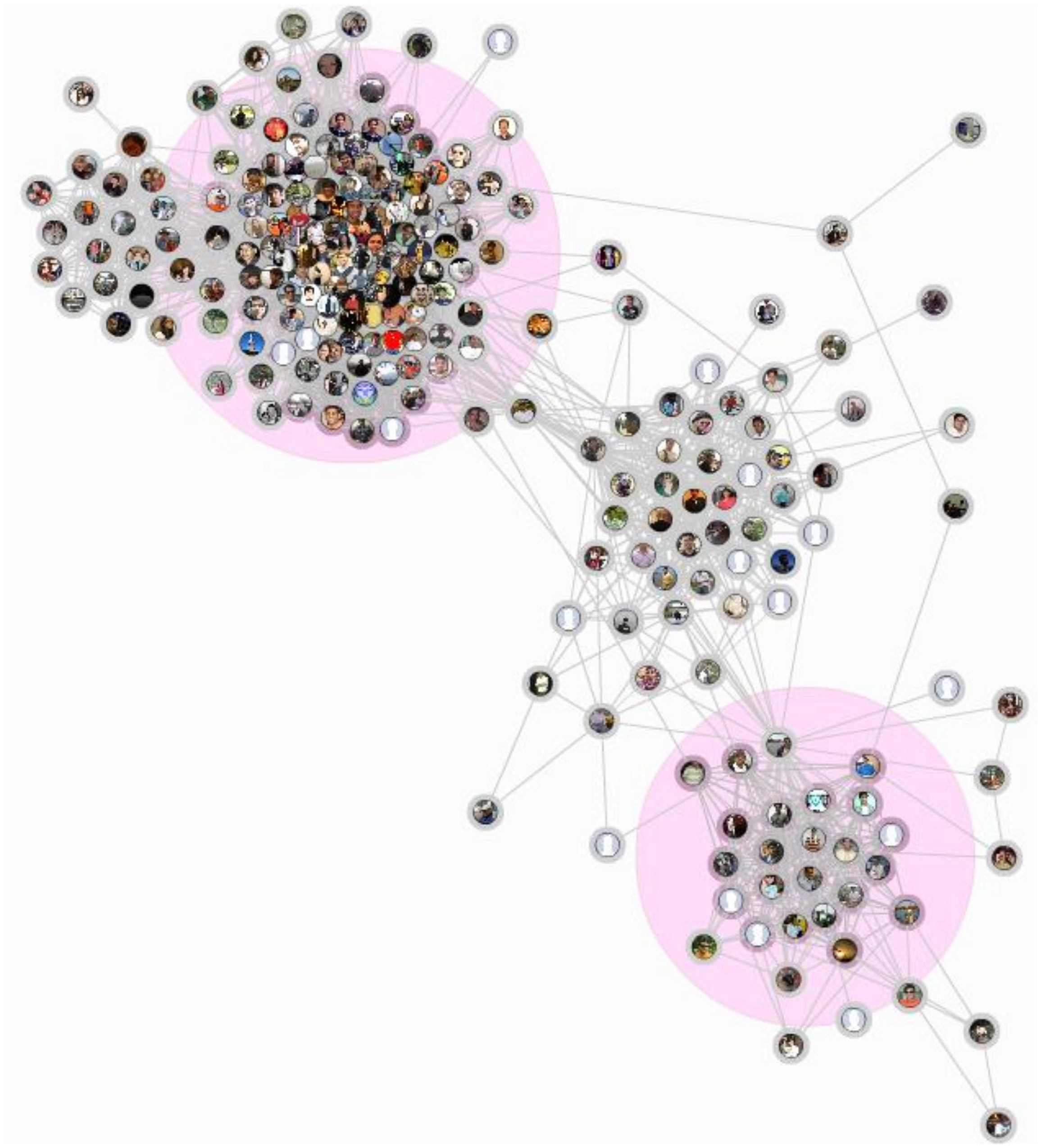

Figure 1. Illustration of cohesive subgroups

Let N be the total number of stocks in consideration and the correlation threshold C_{ij} ($-1 \leq C_{ij} \leq 1$) is calculated using following formula [6, 7]:

$$C_{ij} = \frac{\langle R_i R_j \rangle - \langle R_i \rangle \langle R_j \rangle}{\sqrt{\langle R_i{}^2 - \langle R_i \rangle^2 \rangle \langle R_j{}^2 - \langle R_j \rangle^2 \rangle}} \quad \forall (i, j) \in E$$

Where,

$i = 1,\ldots, N, j = 1,..., N$

P_i(t) is the price of the stock *i* on day t

R_i(t) = $\ln \frac{P_i(t)}{P_i(t-1)}$ defines return or trading volume of the stock *i* on day t

$\langle R_i \rangle = \frac{1}{T} \sum_{t=1}^{T} R_i(t)$ defines mean of R_i(t) over *T* period of time

Different correlation threshold values are used to construct different instances of the stock market graphs depending on the degree of diversification (minimize potential risk measured as correlation associated in the portfolio) needed by user. This example is a graph-based data mining application in stock market. More information about construction of the market graph can be found in [4-6, 8]. Many network models that can be used to solve this diversified portfolio selection problem in stock market were studied and analyzed in literature.

An *independent set* is a set of pair-wise non-adjacent vertices in the graph. A clique is a complete subgraph with every vertex is adjacent to every other vertex in a graph. Independent set and clique are equivalent under graph complementation. Clique was the earliest graph model for representing cohesive subgroups ("*strongly connected*" subgroups in social network) [3, 9]. Cliques have highest possible degree, fastest reachability and strongest connectivity among members. Although, cliques were thought to be perfect to characterize cohesive subgroups, cliques were found to be too idealized and restrictive in practice. In the aforementioned data mining application in stock market, independent sets are used to find a "*perfectly diversified*" portfolio where market graph is generated with a negative correlation threshold θ. Moreover, cliques are used to find group of stocks in pair-wise correlation with each other, i.e., stocks exhibit similar price fluctuations. For example, correlation between Toyota and Ford stocks is higher than correlation between Toyota and IBM. It is because Toyota and Ford belong to the same family of the companies, i.e., both belong to the automobile industry. Intuitively, cliques tend to identify entities similar in nature. An independent set represents a portfolio in the market graph where all stocks in the independent set are negatively correlated. However, choosing such a "*perfectly diversified*" portfolio with a large group of negatively correlated stocks is a difficult task [6]. This idea motivates the relaxation of the independent set by allowing user controlled interdependency in the subgroup where degree of the nodes are bounded from above.

We are motivated by this application and identified relaxations of such models. This relaxation allows finding larger portfolios without losing diversity of the portfolios. Structural relaxations for cliques and independent sets were introduced to explore and analyze more areas within the graph. The clique relaxations such as k-clique (pair-wise distance bounded network) [10-12], k-club (diameter bounded network) [10, 11] and k-plex (degree bounded network) [13] and independent set relaxations such as co-k-plexes [9, 14] were introduced due to practical needs in most of the data mining applications.

This thesis deals with such a graph theoretic independent set relaxation model introduced by Balasundaram in [14] called a *co-k-plex*. The motivation behind this model is that the independent set has no edge between any two vertices in a set whereas a co-k-plex allows a limited number incident at each node. This parameterized model represents an independent set when parameter $k = 1$ and provides an independent set relaxation for $k > 1$. As increasing the value of k corresponds to a gradual relaxation of independence.

This thesis addresses the optimization problem to find a largest co-k-plex in a graph and develops a metaheuristic approach to solve this problem. We focus only on co-k-plex models for parameter k = 1, 2, 3 and the related optimization problems. However, the approach is applicable to any positive integer k.

The organization of this thesis in subsequent chapters is as follows: Chapter II presents required notations and definitions from graph theory; and a review of literature on cliques and clique relaxation models, independent sets and the associated optimization problems. Further, we describe the independent set relaxation model and the associated research problems. Chapter III describes the need for modern metaheuristic algorithms for solving combinatorial optimization problems. One such heuristic algorithm to solve this research problem is then studied in subsequent sections of Chapter III. Numerical results from computational experimentations on a large test-bed of benchmark instances are presented in Chapter IV. Chapter V concludes with important findings and provides quick insight of research work.

CHAPTER II

THE MAXIMUM CO-K-PLEX PROBLEM

2.1 DEFINITIONS, NOTATIONS AND BACKGROUND

Consider a simple, finite and undirected graph G = (V, E), where V is the set of vertices, and E is the set of edges between the vertices; $E \subseteq \{\{u, v\} \mid u, v \in V\}$. Let $\deg_G(v) = |\{u : (u, v) \in E\}|$ denote the degree of v in G; $d_G(u, v)$ denotes the length of a shortest path in terms of number of edges between vertices u and v in G, and $\mathrm{diam}(G) = \max \{d_G(u, v): \forall\ (u, v) \in E\}$ is the diameter of graph G. $G[S] = (S, E \cap (S \times S))$, denotes the subgraph induced by $S \subseteq V$. $N(v)$ denotes the set of vertices which are adjacent to v in graph G, i.e. the number of edges incident at v. A complement graph $\overline{G} = (V, \overline{E})$ of the graph G is a graph with the same vertices as G and with the property that two vertices in $\overline{G}$ are adjacent if and only if they are not adjacent in G, i.e. $e \in \overline{E}$ if and only if $e \notin E$. A graph G is complete graph if its vertices are pair-wise adjacent, i.e. $\forall\ i, j \in V$, $\{i, j\} \in E$. A subset of vertices S of graph G is a clique if, there is an edge between each pair of vertices, i.e., a clique induces a complete subgraph. A clique is called a *maximal* clique if it is not contained in a larger clique. A maximum clique in a graph G is a clique of maximum cardinality in G and the clique number for G denotes by $\omega(G)$ is the cardinality of a maximum clique. The *maximum clique problem* (MCP) is to find a largest clique in the given graph G and it is known to be NP-hard [15]. The *maximum weighted clique problem* (MWCP) (each vertex is associated with positive weight in G) is to find a clique of maximum weight in G. The maximum clique problem is the special case in which all weights are unity.

An independent set is a subset of pair-wise non adjacent vertices in G. $S \subseteq V$ is an independent set in G if and only if it forms a clique in $\overline{G}$. An independent set is said to be *maximal* if it is not contained in a larger independent set. The *maximum independent set problem* (MISP) is to find an independent set of maximum cardinality and the independence number; $\alpha(G)$, is the cardinality of a maximum independent set in G. MISP is known to be NP-hard [15]. The *maximum weighted independent set problem* (MWISP) is to find an independent set of maximum weight in G. Finding a maximum independent set in G is equivalent to finding a maximum clique in $\overline{G}$. The maximum independent set and maximum clique problems are closely related, we deal with both the problems while describing the properties and algorithms for the maximum independent set problem. In this case, it is clear that a result holding for the maximum clique problem in G will also be true for the maximum independent set problem in $\overline{G}$. Thus, researchers may not differentiate these problems, as solving one problem is equivalent to solving the other problem on general graphs. The first algorithm for finding large cliques in a graph was published by Harary and Ross in [16]. Since, many algorithms have been developed to find maximal cliques in the graph [17]. Many exact algorithms based on branch and bound method for maximum clique problem and independent set problem are available in literature. One of the important algorithms is a recursive algorithm for maximum independent set problem proposed by Tarjan and Trojanowski in [18] with complexity of $O(2^{n/3})$ which is then modified to complexity of $O(2^{0.276n})$ by Robson in [19]. Readers are referred to the text in [6, 17] for details regarding of algorithms to find maximal or maximum cliques and related problems in G. Also, many heuristics are available to solve maximum clique and independent set problems in the literature [6]. Chapter III will discuss in detail some greedy heuristics for these problems. The maximum independent set/clique problem has applications in numerous fields, including information retrieval, signal transmission analysis, classification theory, economics, scheduling, and biomedical engineering, experimental design and computer vision and pattern recognition, map labeling and market graph (portfolio optimization) etc. [17].

A subset of vertices S of a graph G is a *k-plex* if, $\deg_{G[S]}(v) = | N(v) \cap S | \geq |S| - k; \forall v \in S$. In other words, a subset of vertices of a G is said to be k-plex if the degree of every vertex in the induced subgraph G[S] is at least |S| - k. A k-plex is *maximal*, if it is not contained in a larger k-plex. Not that a 1-plex is a clique. A maximum k-plex is a k-plex of maximum cardinality in G and the k-plex number for G is denoted as $\omega_k(G)$. The *maximum k-plex problem* is to find a largest k-plex for the given graph and it is known to be NP-hard [9]. The *maximum weighted k-plex problem* is to find a k-plex of maximum weight in G.

A subset of vertices S of a graph G = (V, E) a *co-k-plex* if, $\deg_{G[S]} (v) = | N(v) \cap S | \leq k-1$ $\forall v \in S$. A co-k-plex induces a subgraph in which degree of a vertex $v \in S$ in G[S] is at the most k-1, i.e. G[S] has maximum degree of k-1 or less. Figure 2 illustrates co-k-plexes for integer k =1, 2 and 3. A co-1-plex set in a graph is simply collection of nonadjacent vertices, i.e., an independent set. Furthermore, a co-2-plex set is a set of independent vertices and matched pairs. A co-3-plex set is a set of independent vertices, paths and cycles. Thus, co-k-plexes provide systematic step-by-step relaxations of independent sets for each positive integer k. A co-k-plex is *maximal*, if it is not contained in a larger co-k-plex. The *maximum co-k-plex problem* (MCPP-k) is to find a maximum cardinality co-k-plex in G and the co-k-plex number, $\alpha_k(G)$ is the cardinality of a maximum co-k-plex in G. The maximum co-k-plex problem is also known to be NP-hard [9, 20, 21]. The *maximum weight co-k-plex problem* (MWCPP-k) is to find a co-k-plex of maximum weight in G. Seidman and Foster introduced the k-plex model to find degree bounded cohesive subgraphs in [13]. The k-plexes are cohesive subgraphs which were introduced to relax the structure of cliques. The co-k-plex and k-plex are equivalent under graph complementation, i.e.

the set S is a co-k-plex in G if and only if S is a k-plex in $\overline{G}$. Consequently, the maximum co-k-plex problem and maximum k-plex problem are closely related and equivalent under graph comp-

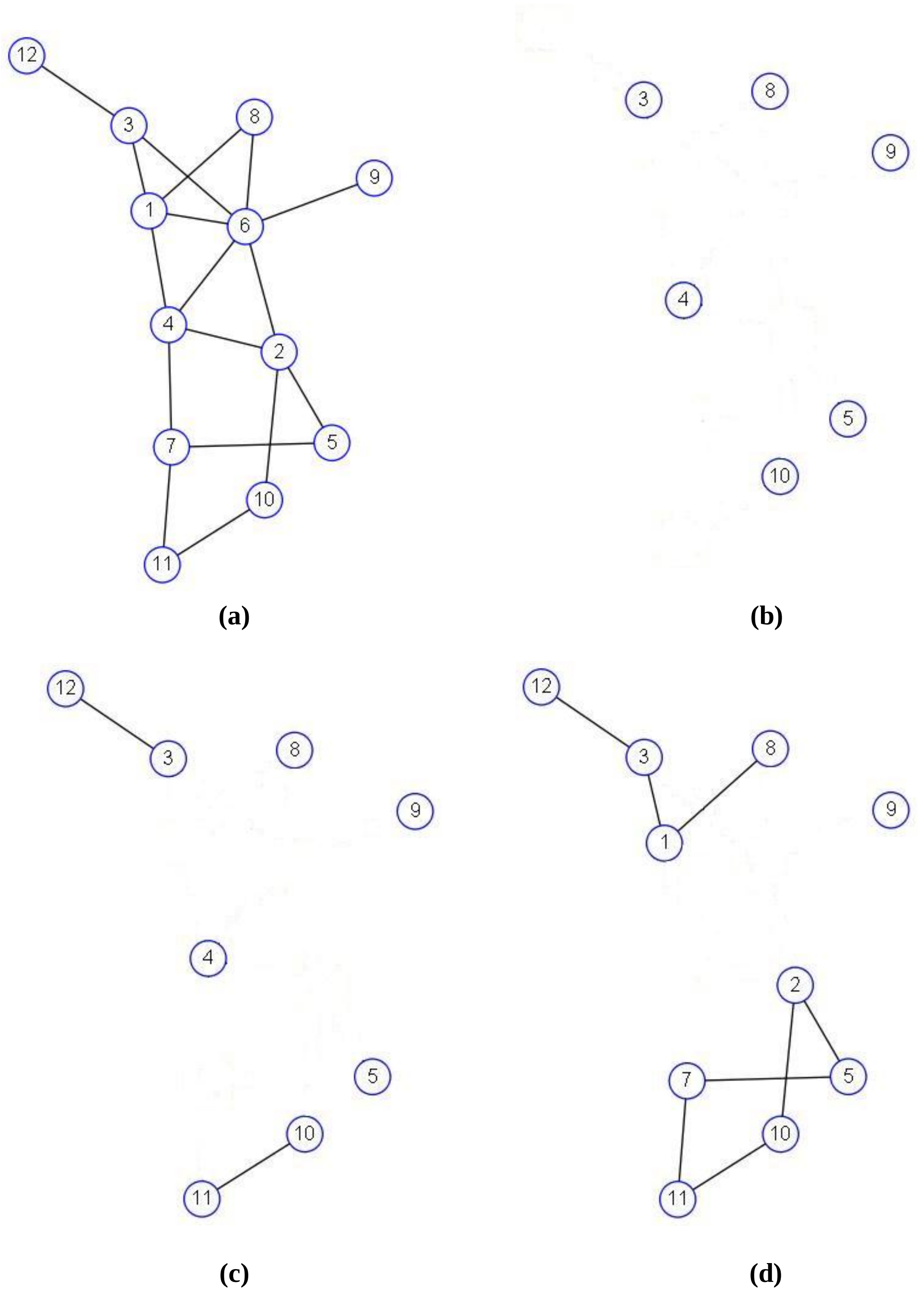

Figure 2. Illustration of co-k-plexes for positive integer k=1, 2, and 3
(a) Original Graph, (b) co-1-plex (subgraph contains independent vertices, i.e., an independent set), (c) co-2-plex (subgraph contains independent vertices and matched pairs), (d) co-3-plex (subgraph contains independent vertices, paths and cycles)

lementation. This is analogous to the relationship between stable sets in G and cliques in $\overline{G}$. Several exact algorithms for these problems are studied in [9, 14, 22, 23]. No metaheuristic algorithm is available to solve this problem presently.

Diversified portfolio solution problem discussed in Chapter I is further studied in perspective of finding diversified portfolios that have superior profitability in [22]. The main drawback of the approach mentioned in Chapter I is the fact that the returns of the identified diversified portfolios were not explicitly taken into account. To addressed this issue of finding large, high-return diversified portfolios into a weighted market graph, each vertex is assigned a weight. The weight of each vertex represents the return of corresponding stock over specified period of time and it can be calculated by two equations as follows [22]:

1) $w_i^1 = \log \frac{P_i(T)}{P_i(1)}$,

2) $w_i^2 = \frac{P_i(T)}{P_i(1)} - 1$,

Where, w_i is the weight of the vertex *i,* i.e., overall return on *i* over *T* trading days. This problem is solved and studied by using maximum weighted clique and k-plex in [22]. It is shown that weighted clique relaxations offer more robustness and profitability as compare to weighted cliques in the context of portfolio selection. It is shown that, 2-plexes offer reasonable "*tight*" clique relaxations by providing good balance between the quality and size of the identified diversified portfolio. This result forms the basis for studying the maximum weighted co-k-plex problem in the context of identifying large, highly diversified profitable portfolios.

2.2. CONTRIBUTIONS

This thesis makes the following contributions. The metaheuristic approach known as *Greedy Randomized Adaptive Search Procedure* (GRASP) has been thoroughly studied and a

GRASP algorithm has been developed for solving the maximum co-k-plex problem and the maximum weighted co-k-plex problem. A suitable neighborhood definition is designed to enhance local search effectiveness, ultimately improving GRASP performance. GRASP algorithms for both problems have been implemented using the C++ programming language. Efficient data structures and implementation techniques are used to reduce GRASP total running time. Benchmark clique instances from the Second DIMACS challenge [9, 14, 24, 25] are complemented to form independent set instances. Computational experiments are conducted on these instances to analyze the effectiveness of GRASP by comparing numerical results of GRASP with results obtained by an exact branch-and-cut algorithm from literature.

CHAPTER III

GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE (GRASP)

3.1 INTRODUCTION TO METAHEURISTICS

Optimization problems of both practical and theoretical importance are divided into two categories: those with continuous variables and those with discrete variables. Combinatorial Optimization (CO) problems fall into a category of problems with discrete values for feasible solutions. In this class of problems, we are looking for an object from a finite (or possibly countably infinite) set of objects that corresponds to the maximum or minimum objective value. Some well-known examples of CO problems are the Traveling Salesman Problem (TSP), the Quadratic Assignment Problem (QAP), and the maximum clique/independent set problems.

A combinatorial optimization (CO) problem is described by a pair (S,f) where$f$ is an objective function to be minimized or maximized as per problem definition and S is a feasible search space; each element of S can be a candidate solution. To solve a combinatorial optimization problem, we need to find a solution s* ∈ S with minimum or maximum objective function value. i.e. f(s*) ≤ f(s) for a minimization problem and f(s*) ≥ f(s) for a maximization problem ∀ s ∈ S. Such a solution s* is a global optimal solution of the CO problem. To solve such problems, exact and heuristic algorithms have been developed and studied. Exact algorithms are guaranteed to provide an optimal solution in finite steps for every finite size instance of a CO problem.

Most of the CO problems are categorized as NP-hard (no polynomial time exists to solve them, unless $P \neq NP$). Therefore, exact algorithms might need exponential time in the worst-case. Heuristic algorithms try to find a good quality solution instead of finding an optimal solution in significantly reduced amount of time. Basic heuristic algorithms are distinguished as constructive and local search methods. Constructive algorithms generate a feasible solution starting from an empty solution. Local search algorithms iteratively find a better quality solution to replace the current solution through intensive exploration of search space based upon appropriately defined neighborhood structures.

A *neighborhood structure* is a mapping $\mathcal{N}$: S$\rightarrow$ 2^S that assigns to every s $\in$ S set of neighbors $\mathcal{N}$(s) $\subseteq$ S. $\mathcal{N}$(s) is called the neighborhood of s. A *locally minimal* solution with respect to a neighborhood structure $\mathcal{N}$ is a solution ŝ such that $\forall$ s $\in \mathcal{N}$(ŝ), f(ŝ) $\leq f$(s). We call ŝ a strict locally minimal solution if f(ŝ) $< f$(s) $\forall$ s $\in \mathcal{N}$(ŝ), s $\neq$ ŝ.

Metaheuristics are widely used to solve combinatorial optimization problems because of their simplicity and robustness. Metaheuristics are designed to produce good quality solutions in reduced amount of time for complex combinatorial problems. Metaheuristics are not problem specific and provide a framework for finding good quality solutions by efficiently exploring neighborhood in short amount of computational time. Simulated Annealing (SA) [26], Tabu Search (TS) [27-29], Greedy Randomized Adaptive Search Procedure (GRASP) [30, 31], Genetic Algorithms (GA) [32, 33] and Variable Neighborhood Search (VNS) [34] are some of the well-known metaheuristics used to solve large CO problems. Performance of metaheuristic algorithms depends on the efficiency of algorithms to explore and exploit the search space. Therefore every metaheuristic algorithm should be carefully designed with such aim.

Intensification and *Diversification* (I&D) are the most powerful and effective factors driving metaheuristics performance to high levels. Glover and Laguna provide high level descriptions of intensification and diversification in [27]. During intensification, algorithm explores search space for high quality solutions in the neighborhood and during diversification, moves to unexplored search space are made to find better solutions than found during intensification. The moves in the search space greatly depend on the search philosophy of a metaheuristic itself and the neighborhood structures defined for the problem. Therefore, one has to ensure right balance between intensification and diversification to obtain an effective metaheuristic.

3.2 GRASP METAHEURISTIC

GRASP is a multistart metaheuristic used to solve combinatorial optimization problems [30, 31, 35, 36]. An iteration of the GRASP consists of two phases: a greedy randomized construction phase and a local search phase. During the construction phase, an initial feasible solution is built and in the local search phase a local optimum is found by investigating the neighborhood of the current feasible solution based on neighborhood definition. The best solution encountered during different iterations is returned by GRASP as final solution. GRASP is the first metaheuristic that was built to construct initial solution by balancing greedy and randomized approaches to enhance the performance of the local search. Hence, GRASP solutions are generally significantly better than the single solution obtained from a random starting point. GRASP has been successfully applied to a wide range of combinatorial optimization problems, such as drawing and turbine balancing, scheduling, routing , problems in logic, including SAT, MAX-SAT, and logical clause inference, partitioning problems, location and layout problems, Graph theoretic applications, assignment problems, transportation problems etc. An annotated bibliography of the GRASP literature from 1989 to 2008 is available in [37, 38].

The maximum co-k-plex problem is a NP-hard problem as discussed in Chapter II. Therefore, metaheuristic algorithms can provide a good approach to solve the maximum co-k-plex problem in relatively small amount of time to produce good quality solutions where exact algorithms might take high computational time in the worst-case. The overall goal of this thesis is to develop GRASP algorithm to solve MCPP-k and MWCPP-k on large graphs. This is the first metaheuristic approach developed for this problem. The next section describes a generic GRASP metaheuristic framework in the context of finding maximum independent sets in a given graph developed by Resende et al. in [39]. Subsequently, we describe the GRASP metaheuristic algorithm for the maximum co-k-plex and the maximum weighted co-k-plex problems developed as part of this thesis research.

3.2.1 GRASP FOR THE MAXIMUM INDEPENDENT SET PROBLEM

Figure 3 and 4 present the pseudo-codes of a generic GRASP and the GRASP construction procedure for the maximum independent set problem, respectively [39]. The parameters Max_Iterations and Seed are the maximum number of iterations for GRASP termination and the initial seed for the pseudorandom number generator, respectively. Construction phase starts with an empty solution and the ordered candidate list (contains elements that can be added one at a time to the current solution without violating feasibility) which contains all the vertices. The list of the best candidates, called a *Restricted Candidate List* (RCL) is constructed choosing best candidates from the candidate list by a greedy-randomized function which depends on a parameter α ($0 \leq \alpha \leq 1$).

```
GRASP(Max_Iterations,Seed)
Read_Input (V, E);
Initialize Best Solution $I^*$ to an empty set;
for $i = 1, \ldots,$ Max_Iterations do
    Generate $\alpha \in [0, 1]$ randomly;
    $I^0$ = GreedyRandomizedConstruction($\alpha$);
    $I$ = LocalSearch($I^0$);
    if $|I| > |I^*|$ do
        $I^* = I$;
    end if
end for
return $I^*$;
end GRASP
```

Figure 3. Pseudo-code of generic GRASP for the maximum independent set problem

The value of parameter α decides the degree of "*greediness*" and "*randomness*" in the algorithm. α = 1 provides pure randomness and α = 0 provides pure greediness. An initial feasible solution is built iteratively by randomly selecting element from RCL and adding to the current solution one at a time. The technique of choosing elements randomly may generate different solutions at each GRASP iteration and the value of α is also chosen at random (but fixed for each major GRASP iteration). At each construction iteration the candidate list is updated and then ordered by using the greedy rule. The construction phase terminates when the candidate list is empty, i.e. no candidate is available to improve current solution. Let us consider the graph $G = (V, E)$ as shown in Figure 2 (a) to solve the MISP.

Let us consider α is generated equal to 0.5 in an iteration of the GRASP. The construction phase starts with an empty solution ($I = \{\}$) and builds an independent set by adding vertices one by one from RCL. C initially consists all the vertices, i.e. $C = V$ and all the elements are ranked in descending order based on degrees of the elements in $G[C]$. (RCL) is constructed by adding vertices which have a lesser or equal degree than threshold (Threshold Value = $dmin + \alpha\,(dmax– dmin)$) in $G[C]$. An element is chosen randomly from RCL, suppose we have chosen vertex 3 (I = {3}).

```
GreedyRandomizedConstruction ($\alpha$)
Initial Independent set, $I^0 = \emptyset$;
Candidate list, $C = V$;
while$|C| > 0$ do
    Let $G[C]$ be the subgraph induced by the vertices in $C$;
    Let $\deg_{G(C)}(u)$ be the degree of $u \in C$ with respect to $G[C]$;
    $dmin = \min\{\deg_{G(C)}(u) \mid u \in C\}$;
    $dmax = \max\{\deg_{G(C)}(u) \mid u \in C\}$;
    RCL = $\{u \in C \mid \deg_{G(C)}(u) \leq dmin + \alpha\,(dmax - dmin)\}$;
    Select $u$ at random from the RCL;
    $I^0 = I^0 \cup \{u\}$;
    $C = C \setminus (N(u) \cup \{u\})$;
end while;
return $I^0$;
end GreedyRandomizedConstruction;
```

Figure 4. Pseudo-code of the GRASP construction procedure for the MISP

Thus, GRASP uses vertex degrees as the greedy function for constructing an independent set. After adding a randomly selected vertex, *C* is updated such that if elements in *C* can be added to current independent set one at a time without violating feasibility, i.e. selecting elements from vertex list that are nonadjacent to every element in *I*. Thus, vertices 2, 4, 5, 7, 8, 9, 10 and 11 forms updated candidate list. The steps discussed so far are repeated until the termination condition (the candidate list is not empty) of construction phase is satisfied.

C	d_{min}	d_{max}	Threshold	RCL	Solution
{1,2,3,4,5,6,7,8,9,10,11,12}	1	6	3.5	{3,5,7,8,9,10,11,12}	{3}
{2,4,5,7,8,9,10,11}	0	3	1.5	{8,9}	{3,8}
{2,4,5,7,9,10,11}	0	3	1.5	{9}	{3,8,9}
{2,4,5,7,10,11}	2	3	2.5	{4,5,10,11}	{3,4,8,9}
{5,10,11}	0	1	0.5	{5}	{3,4,5,8,9}
{10,11}	1	1	1	{10}	{3,4,5,8,9,10}

Table 1. Iterative greedy randomized construction phase for the MISP

Table 1 presents the steps carried out in the greedy randomized construction phase to construct an initial feasible solution. The greedy randomized construction phase for the MISP returns an initial feasible solution, i.e. Solution set $I = \{3, 4, 5, 8, 9, 10\}$ as shown in Figure 2 (b).

Figure 5 shows the pseudo-code of the local search phase for the maximum independent set problem. Applying local search to improve each constructed solution is beneficial because of the fact that constructed solutions are not guaranteed to be globally optimal especially for NP-hard CO problems. The GRASP local search iteratively replaces the current solution by a better solution from the neighborhood and terminates when no better solution is found in the neighborhood. Thus, the performance of the local search algorithm depends on three factors: the choice of neighborhood structure, efficient neighborhood search techniques and initial feasible solution constructed in the construction phase. One possible neighborhood structure for local search can be implemented by (2, 1) swap or exchange. This approach adds two new elements to current solution after removing one element. The local search terminates when no such triplet can be found and returns the current solution as the local optimum. Given a (maximal) independent set (I), the (2,1)-exchange local search neighborhood is given by $\mathcal{N}_2$(I) = { J ⊆ V : J is an independent set, |I \ J| = 1 and |J \ I| = 2}. Local search starts with initial feasible solution, I^0. Suppose, initial feasible solution constructed during greedy randomized construction phase is I^0 = {5, 6, 10, 12}. Note that I^0 is a maximal independent set. Local search tries to find a triplet in the neighborhood of current solution. Table 2 presents solutions found during an iterative local search phase, *I* = {3, 4, 5, 8, 9, 10} as shown in Figure 2 (b).

```
LocalSearch(I)
H(I) = {(v, u, w) | v, u ∈V \ I, (v, u) ∉ E, w ∈ I, and
        v and u are nonadjacent to all vertices of I except w};
while |H(I)| > 0 do
        Select (u, v, w) ∈ H(I);
        I = I ∪{u, v} \ {w};
        H(I) = {(v, u, w) | v, u ∈ V \ I, (v, u) ∉ E, w ∈ I, and
                v and u are nonadjacent to all vertices of I except w};
end while
return I;
end LocalSearch
```

Figure 5. Pseudo-code of (2, 1)-exchange local search procedure for the MISP

H(*I*)	Solution
{8,9,6}	{5,8,9,10,12}
{3,4,12}	{3,4,5,8,9,10}

Table 2. Local search phase for the MISP

The local search algorithm terminates when *H*(*I*) is empty, i.e. current solution cannot be further improved. Thus, current independent set solution is said to be locally optimal and it is returned to the GRASP where it is compared to the best independent set solution found so far. If the locally optimal solution has higher cardinality than the best solution, then it is replaced by the current locally optimal solution. In the next section, we will design GRASP in the context of co-k-plexes for the maximum co-k-plex problem in G.

3.2.2 GRASP FOR THE MAXIMUM CO-K-PLEX PROBLEM

The pseudo-code of the generic GRASP and the greedy randomized construction phase for the maximum co-k-plex problem are similar to that of for the maximum independent set problem. After each GRASP iteration, locally maximum co-k-plex ($\mathbb{C}$) solution is compared with best co-k-plex ($\mathbb{C}^*$) solution found so far. The best co-k-plex solution is replaced by the current locally maximum co-k-plex solution if it is larger ($|\mathbb{C}| > |\mathbb{C}^*|$). Finally, the best co-k-plex solution encountered during different iterations is returned by GRASP.

```
GreedyRandomizedConstruction ($\alpha$)
Initial co-k-plex, $\mathbb{C}^0 = \emptyset$;
Candidate list, $C$ = V;
while $|C| > 0$ do
        Let G[$C$] be the subgraph induced by the vertices in $C$;
        Let $\deg_{G(C)}(u)$ be the degree of $u \in C$ with respect to G[$C$];
        $d_{min} = \min\{\deg_{G(C)}(u) \mid u \in C\}$;
        $d_{max} = \max\{\deg_{G(C)}(u) \mid u \in C\}$;
        RCL = $\{u \in C \mid \deg_{G(C)}(u) \le d_{min} + \alpha (d_{max} - d_{min})\}$;
        Select $u$ at random from the RCL;
        $\mathbb{C}^0 = \mathbb{C}^0 \cup \{u\}$;
        $C = \{u \in C : |N(u) \cap \mathbb{C}^0| \le k\text{-}1 \text{ and } N(u) \cap S = \emptyset\}$,
             where, $S = \{u \in \mathbb{C}^0 : |N(u) \cap \mathbb{C}^0| = k\text{-}1\}$;
end while;
return $\mathbb{C}^0$;
end GreedyRandomizedConstruction;
```

Figure 6. Pseudo-code of greedy randomized construction for the MCPP-k

During the greedy randomized construction phase for the MCPP-k, candidate list C is updated such that $C = \{u \in C : |N(u) \cap \mathbb{C}| \le k\text{-}1 \text{ and } N(u) \cap S = \emptyset\}$ where, $S = \{u \in \mathbb{C} : |N(u) \cap \mathbb{C}| = k\text{-}1\}$, i.e., u can be added in C if it has no more than k-1 neighbors in the current co-k-plex and no saturated node in S (node at degree k-1 in the current co-k-plex) is adjacent to u. Let us consider the graph G = (V, E) shown in Figure 2 (a) to solve the maximum co-k-plex problem for k = 2 (co-2-plex). Suppose parameter α is 0.5 in the following numerical example. Table 3 presents the steps carried out in the greedy randomized construction phase for the MCPP-2 to construct an initial feasible solution. The greedy randomized construction phase for the MCPP-2 returns an initial feasible solution, i.e. Solution set $\mathbb{C}$ = {3, 4, 5, 8, 9, 10, 11, 12} as shown in Figure 2 (c).

C	d_{min}	d_{max}	Threshold	RCL	Co-2-plex
{1,2,3,4,5,6,7,8,9,10,11,12}	1	6	3.5	{3,5,7,8,9,10,11,12}	{3}
{1,2,4,5,6,7,8,9,10,11,12}	0	5	2.5	{5,8,9,10,11,12}	{3,8}
{2,4,5,7,9,10,11,12}	0	3	1.5	{9}	{3,8,9}
{2,4,5,7,10,11,12}	0	3	1.5	{12}	{3,8,9,12}
{2,4,5,7,10,11}	2	3	2.5	{7,11}	{3,8,9,11,12}
{2,4,5,7,10}	1	3	2	{4,5,7,10}	{3,8,9,10,11,12}
{4,5}	0	0	0	{4,5}	{3,4,8,9,10,11,12}
{5}	0	0	0	{5}	{3,4,5,8,9,10,11,12}

Table 3. Iterative greedy randomized construction phase for the MCPP-2

The main difference between local search for MCPP-k and MISP is in the neighborhood definition. The local search neighborhood for MCPP-k is defined as, for a given (maximal) co-k-plex ($\mathbb{C}$), $\mathcal{N}_q(\mathbb{C})$ = { J ⊆ V : J is a co-k-plex and | $\mathbb{C}$ \ J | = 1 and | J \ $\mathbb{C}$ | = q}, i.e. (q, 1) exchange neighborhood for q = 2, 3, 4…. The neighborhood search can be implemented by using either a best-improving or first improving move strategy. In best–improving strategy, all neighbors of the current solution are investigated and evaluated for best improving neighbor. But first improving strategy uses the first improving neighbor encountered. We use the first improving strategy to reduce the running time of algorithm.

$H(\mathbb{C})$	Co-2-plex
{8,12,3}	{6,7,8,11,12}
{4,5,10,7}	{4,5,6,8,10,11,12}
{3,9,6}	{3,4,5,8,9,10,11,12}

Table 4. Local search phase for the MCPP-2

Local search starts with the initial feasible solution constructed during the greedy randomized construction phase. Suppose, the initial feasible solution constructed during greedy randomized construction phase is $\mathbb{C}^0 = \{3, 6, 7, 11\}$. Table 2 presents local search moves during an iterative local search phase resulting in the local optimum $\mathbb{C} = \{3, 4, 5, 8, 9, 10, 11, 12\}$ as shown in Figure 2 (c).

```
LocalSearch($\mathbb{C}$)
$H(\mathbb{C}) = \{(u_i, \ldots, u_q, w) \mid u_i \in V \setminus \mathbb{C}, w \in \mathbb{C}\ \forall\ i = 1, \ldots, q$, and
                    $\mid \mathbb{C} = \mathbb{C} \setminus \{w\} \cup \{u_{i, \ldots,} u_q\}$ is a co-k-plex$\}$
while $|H(\mathbb{C})| > 0$ do
        Select $(u_i, \ldots, u_q, w) \in H(\mathbb{C})$;
        $\mathbb{C} = \mathbb{C} \cup \{u_i, \ldots, u_q\} \setminus w$;
        $H(\mathbb{C}) = \{(u_i, \ldots, u_q, w) \mid u_i \in V \setminus \mathbb{C}, w \in \mathbb{C}\ \forall\ i = 1, \ldots, q$, and
                    $\mid \mathbb{C} = \mathbb{C} \setminus \{w\} \cup \{u_{i, \ldots,} u_q\}$ is a co-k-plex$\}$
end while
return $\mathbb{C}$;
end LocalSearch
```

Figure 7. Pseudo-code of local search for the MCPP-k

3.2.3 GRASP FOR THE MAXIMUM WEIGHTED CO-K-PLEX PROBLEM

The pseudo-code of the GRASP for the maximum weighted co-k-plex problem is similar to that of the maximum independent set problem. After each GRASP iteration, locally maximum weighted co-k-plex ($\mathbb{C}_w$) solution is compared with best weighted co-k-plex ($\mathbb{C}_w$*) solution found so far. The best weighted co-k-plex solution is replaced by the current locally maximum weighted co-k-plex solution if it has larger weight ($W(\mathbb{C}_w) > W(\mathbb{C}_w^*)$). Finally, the best weighted co-k-plex solution encountered during different iterations is returned by GRASP. During the greedy randomized construction phase for the MWCPP-k, RCL is constructed by selecting candidates in C such that RCL = $\{u \in C \mid W(u) \geq W_{min} + \alpha\,(W_{max} - W_{min})\}$.

Let us consider the graph G = (V, E) shown in Figure 2 (a) to solve the maximum weighted co-k-plex problem for k = 2 (co-2-plex). Suppose parameter α is 0.5. Table 5 presents weights associated with each vertex in the graph G. Table 6 presents the steps carried out in the greedy randomized construction phase for the MWCPP-2 to construct initial feasible solution.

The greedy randomized construction phase for the MWCPP-2 returns an initial feasible solution, i.e. Solution set $\mathbb{C}_w{}^0 = \{1, 4, 5, 9, 10, 11, 12\}$ with maximum weight, $W(\mathbb{C}_w{}^0) = 460$ units.

```
GreedyRandomizedConstruction ($\alpha$)
Initial weighted co-k-plex, $\mathbb{C}_w{}^0 = \emptyset$;
Candidate list, $C = V$;
while $|C| > 0$ do
        Let $G[C]$ be the subgraph induced by the vertices in $C$;
        Let $W(u)$ be the weight of $u \in C$;
        $W_{min} = \min\{W(u) \mid u \in C\}$;
        $W_{max} = \max\{W(u) \mid u \in C\}$;
        RCL = $\{u \in C \mid W(u) \geq W_{min} + \alpha (W_{max} - W_{min})\}$;
        Select $u$ at random from the RCL;
        $\mathbb{C}_w{}^0 = \mathbb{C}_w{}^0 \cup \{u\}$;
        $C = \{u \in C : |N(u) \cap \mathbb{C}_w{}^0| \leq k\text{-}1$ and $N(u) \cap S = \emptyset\}$,
             where, $S = \{u \in \mathbb{C}_w{}^0 : |N(u) \cap \mathbb{C}_w{}^0| = k\text{-}1\}$;
end while;
return $\mathbb{C}^0$ ;
end GreedyRandomizedConstruction;
```

Figure 8. Pseudo-code of greedy randomized construction for the MWCPP-k

u	1	2	3	4	5	6	7	8	9	10	11	12
$W(u)$	120	50	20	100	30	25	15	35	105	45	55	5

Table 5. Vertex weights in G

C	W_{min}	W_{max}	Threshold	RCL	Co-2-plex
{1,2,3,4,5,6,7,8,9,10,11,12}	5	120	62.5	{1,4,9}	{4}
{1,2,3,5,6,7,8,9,10,11,12}	5	120	62.5	{1,9}	{1,4}
{5,9,10,11,12}	5	105	55	{9,11}	{1,4,9}
{5,10,11,12}	5	55	30	{5,10,11}	{1,4,9,11}
{5,10,12}	5	45	25	{5,10}	{1,4,9,10,11}
{5,12}	5	30	17.5	{5}	{1,4,5,9,10,11}
{12}	5	5	5	{12}	{1,4,5,9,10,11,12}

Table 6. Iterative greedy randomized construction phase for the MWCPP-2

Local search starts with an initial feasible solution constructed during the greedy randomized construction phase. Suppose, initial feasible solution constructed during the greedy randomized construction phase is $\mathbb{C}_w^{\,0} = \{3, 6, 7, 11\}$ with weight, $W(\mathbb{C}_w^{\,0}) = 115$. Table 7 presents local search moves during an iterative local search phase resulting in the local optimum $\mathbb{C}_w = \{3, 4, 5, 8, 9, 10, 11, 12\}$ with weight $W(\mathbb{C}_w) = 395$ units as shown in Figure 2 (c).

```
LocalSearch($\mathbb{C}_w$)
$H(\mathbb{C}_w) = \{(u_i, \ldots, u_q, w) \mid u_i \in V \setminus \mathbb{C}_w, w \in \mathbb{C}_w \ \forall\ i = 1, \ldots, q$, and
                          $\mid \mathbb{C}_w = \mathbb{C}_w \setminus \{w\} \cup \{u_{i,\ldots,} u_q\}$ is a co-k-plex and
                          $\mid \sum W(u_i) > W(w)\}$
while $\mid H(\mathbb{C}_w) \mid > 0$ do
Select $(u_i, \ldots, u_q, w) \in H(\mathbb{C}_w)$;
$\mathbb{C}_w = \mathbb{C}_w \cup \{u_i, \ldots, u_q\} \setminus \{w\}$;
$H(\mathbb{C}_w) = \{(u_i, \ldots, u_q, w) \mid u_i \in V \setminus \mathbb{C}_w, w \in \mathbb{C}_w \ \forall\ i = 1, \ldots, q$, and
                          $\mid \mathbb{C}_w = \mathbb{C}_w \setminus \{w\} \cup \{u_{i,\ldots,} u_q\}$ is a co-k-plex and
                          $\mid \sum W(u_i) > W(w)\}$
end while
return $\mathbb{C}_w$;
end LocalSearch
```

Figure 9. Pseudo-code of local search for the MWCPP-k

$H(\mathbb{C}_w)$	Co-2-plex	Weight
{2,8,9,12,6}	{2,3,7,8,9,11,12}	285
{4,7}	{2,3,4,8,9,11,12}	370
{5,10,2}	{3,4,5,8,9,10,11,12}	395

Table 7. Local search phase for the MWCPP-2

In Chapter IV, extensive computational experiments are conducted to test the effectiveness of both versions of GRASP for the MCPP-k and the MWCPP-k proposed in this chapter.

CHAPTER IV

COMPUTATIONAL EXPERIMENTS

This chapter presents our computational experiments conducted on both versions of GRASP algorithm developed in Chapter III. Extensive investigation of GRASP algorithms proposed for the maximum co-k-plex and weighted co-k-plex problems are performed on a large test-bed of benchmark instances. In Section 4.1, we describe all the relevant GRASP algorithm implementation details and in Section 4.2 we describe the instances used in testing. Our results on both versions of GRASP algorithm are presented in Section 4.3.

4.1 GENERAL IMPLEMENTATION DETAILS

This section describes implementation details that are common to all our experiments. All numerical experiments were conducted on Dell Precision T3500 with system configuration as follows: Intel®Xeon®W3550 3.07 GHz processor, 3GB RAM and Microsoft Windows XP Professional Version 2002 Service Pack 3 operating system. Two versions of the GRASP algorithm for solving MCPP-k and MWCPP-k are implemented using the C++ programming language. We use an adjacency list to represent graphs. An adjacency list contains lists of all vertices that are adjacent to each vertex in the graph. Table 8 illustrates the adjacency list for the graph in Figure 2 (a).

Vertex	Adjacent vertices					
1	3	4	6	8		
2	4	5	6	10		
3	1	6	12			
4	1	2	6	7		
5	2	7				
6	1	2	3	4	8	9
7	4	5	7			
8	1	6				
9	6					
10	2	11				
11	7	10				
12	3					

Table 8. Illustration of representation of vertex adjacency list

While updating candidate list and determining degree of vertex in subgraph induced by elements of candidate list during construction phase of GRASP, we recognized that these steps take considerable amount of time. It is also observed that during local search, enumerating neighboring solutions needs considerable amount of time. Thus, we used two vectors to keep track of vertex degree in subgraph induced by elements of candidate list (C) and vertex degree in subgraph induced by elements in current co-k-plex (solution) respectively as follows:

```
vector<int> DegreeCand;          //keeps track of degree of the node in a subgraph induced
                                 by elements of candidate list

vector<int> AddCandidate;        //keeps track of the degree of each node in current co-k-
                                 plex (solution), i.e. deg_G(ℂ)(u), u ∈V
```

Initially, each element in vector "*DegreeCand*" represents degree of each vertex in original graph. Degree of each vertex adjacent to $u \in C$ is reduced by one when u is selected radomly from RCL to add in the current co-k-plex ($\mathbb{C}$). This facilitates step of updating C by keeping track of degree of v, $v \in C$ & $v \notin \mathbb{C}$, i.e.degree of vertex in G[C]. This also helps to determine maximum and minimum degree in G[C] easily. The vector "*AddCandidate*" keeps track of degree of each vertex in $\mathbb{C}$ (number of vertex adjacent to u in $\mathbb{C}$) by increasing degree of each vertex adjacent to $u \in \mathrm{V}$ when u is added to current $\mathbb{C}$. This helps to determine degree of $v \in \mathrm{V}$ in $\mathrm{G}[\mathbb{C}]$ and facilitates the step of finding improving neighbors quickly. Table 9 illustrates improvement in GRASP runtime. The naive version explicitly checks neighboring solutions while the improved version uses the above mentioned data structures.

Graph	|V|	|E|	Naive GRASP Time (secs)	Improved GRASP Time (secs)
c-fat500-2.clq	500	9139	42.81	0.031
c-fat500-10.clq	500	46627	274.86	0.328
hamming10-2.clq	1024	518656	1504.40	6.937

Table 9. Illustration of improvement in the GRASP running time (Max_Iterations = 10)

4.2 DESCRIPTION OF THE TEST-BED

In our Experiments, we used benchmark clique instances from Second DIMACS challenge [9, 14, 24, 25]. These instances have been used in the literature while studying algorithms for the maximum k-plex problem [9]. Table 10 presents information regarding the 32 DIMACS benchmark instances used in this thesis for experimental purpose [24, 25]. Description of DIMACS instances from Table 10 can be found in [9, 14, 40, 41]. For solving the maximum weighted co-k-plex problem, weights are generated randomly between range 1 to 10n unit (n = vertex set size) for the DIMACS instances.

4.3 NUMERICAL RESULTS: MCPP-k AND MWCPP-k

GRASP algorithms for MCPP-k and MWCPP-k were implemented and studied on instances in Table 10 for k= 1, 2, 3. GRASP results are compared with the results obtained by Branch-and-Cut exact algorithm in [9]. All numerical results in [9] on benchmark clique instances were obtained by conducting experiments on Dell Precision PWS690® computers with 2.66GHz Xeon®

processor, 3GB RAM and 120GB HDD. Branch-and-Cut (BC) algorithm was implemented using ILOG CPLEX 10.0® for solving the maximum k-plex and maximum weighted k-plex problems for k =1, 2, 3 in [9]. Description of the BC algorithm and CPLEX® parameters can be found in [9, 14].

Graphs	|V|	|E|	*edge density*
brock200_1.clq	200	14834	0.745
brock200_2.clq	200	9876	0.496
brock200_4.clq	200	13089	0.658
brock400_2.clq	400	59786	0.749
brock400_4.clq	400	59765	0.749
brock800_2.clq	800	208166	0.651
brock800_4.clq	800	207643	0.65
c-fat200-1.clq	200	1534	0.077
c-fat200-2.clq	200	3235	0.163
c-fat200-5.clq	200	8473	0.426
c-fat500-1.clq	500	4459	0.036
c-fat500-2.clq	500	9139	0.073
c-fat500-5.clq	500	23191	0.186
c-fat500-10.clq	500	46627	0.374
hamming6-2.clq	64	1824	0.905
hamming6-4.clq	64	704	0.349
hamming8-2.clq	256	31616	0.969
hamming8-4.clq	256	20864	0.639
hamming10-2.clq	1024	518656	990
hamming10-4.clq	1024	434176	0.829
johnson8-2-4.clq	28	210	0.556
johnson8-4-4.clq	70	1855	0.768
MANN_a9.clq	45	918	0.927
MANN_a27.clq	378	70551	0.99
MANN_a45.clq	1035	533115	0.996
keller4.clq	171	9435	0.649
p_hat300-1.clq	300	10933	0.244
p_hat300-2.clq	300	21928	0.489
p_hat300-3.clq	300	33390	0.744
p_hat700-1.clq	700	60999	0.249
p_hat700-2.clq	700	121728	0.498
p_hat700-3.clq	700	183010	0.748

Table 10. DIMACS benchmarks

GRASP parameter "*Max_Iterations*" (termination criterion for GRASP algorithms) is set to be 10 (Max_Iterations = 10). It is determined based on preliminary experiments on the GRASP algorithm. Note that increasing GRASP iterations increases the probability of GRASP finding better quality solutions, but it also increases computational time (Appendix A, Table 11). It was found in preliminary experiments that 10 GRASP iterations provide good balance between quality of solutions and amount of computational time required for this test bed of instances. Data collected after termination of GRASP algorithm are as follows:

- $A_k(G)$ = Cardinality of best co-k-plex found by GRASP
- $I_k(G)$ = Cardinality of best weighted co-k-plex found by GRASP
- $\alpha_k^W(G)$ = Best weight of the co-k-plex found by GRASP
- CTime = Construction phase total running time in secs
- LSTime = Local search phase total running time in secs
- GRASPTime ≈ CTime+ LSTime = GRASP total running time in secs
- LSHitRate = Number of times local search improved constructed initial feasible solution
- LSAvgPerInc = Average percentage improvement (size or weight) over initial constructed solution by local search

Appendix A, Table 12 illustrates GRASP performance over BC exact algorithm. BC algorithm is used to find the maximum cardinality k-plex in G where GRASP is used to find maximum cardinality co-k-plex in $\bar{G}$ both problems are equivalent under graph complementation. BC algorithm was run on DIMACS instances by setting a time limit of 10800 secs in [9]. BC algorithm terminates when it hits this time limit and provides a lower and upper bound on the optimum. The lower bound is more appropriate to compare with GRASP solutions. It is clear from the results that GRASP is able to find good quality solutions very quickly.

Appendix A, Table 13 and Table 14 illustrate GRASP performance for solving the maximum weighted co-k-plex problem in $\overline{G}$ over BC algorithm used to solve the maximum weighted k-plex problem in G. BC algorithm was run on DIMACS instances by setting a time limit of 3600 secs. As before GRASP is producing good quality solutions in very little time. Overall GRASP results for the maximum co-k-plex and weighted co-k-plex problems can be found in Appendix B.

CHAPTER V

CONCLUSION AND FUTURE WORK

This thesis presents the first metaheuristic approach to solve the maximum co-k-plex problem. In literature, GRASP has been successfully implemented to solve many hard combinatorial optimization problems such as the maximum clique and independent set problems [6, 39, 42]. The main contribution of this thesis is the development of GRASP metaheuristic to solve the maximum co-k-plex and weighted co-k-plex problems. The algorithmic development of construction and local search phases for solving the maximum co-k-plex and weighted co-k-plex problems were found to be the major challenges. Particularly, updating the candidate list in the construction phase was found to be time consuming. An appropriate data structure and an efficient procedure are designed to effectively update the candidate list that considerably reduced the computational time of the construction phase.

The local search phase is enhanced by designing a suitable neighborhood definition that ultimately improved the performance of GRASP. Further, GRASP developed to solve the maximum weighted co-k-plex problem can be used to find large profitable portfolios of stocks with high diversity. The maximum k-plex problem in G is computationally equivalent to the maximum co-k-plex problem in $\overline{G}$ as discussed in Chapter II. Benchmark clique instances from second DIMACS challenge [9, 14, 24, 25] are used to test the algorithms developed.

Effectiveness of the developed GRASP algorithms is confirmed by conducting extensive computational experiments on these instances. Also, we have compared solutions provided by both GRASP algorithms with results obtained by exact branch-and-cut algorithm from literature [9].

Future Directions for Research. The maximum co-k-plex and weighted co-k-plex problems can be used to find large profitable portfolios. A detailed computational study on very large-scale stock market data can be done for testing effectiveness of the GRASP proposed in this thesis.

In the recent past, hybridization of metaheuristics has emerged as an effective approach to enhance metaheuristics performance. Hybridization of a GRASP with other metaheuristics was successfully done in the literature to create new and effective metaheuristics. Some of the ways to hybridize the GRASP can be found in [43]. For instance, one can try to extend a GRASP algorithm proposed in this thesis to include path-relinking and the local search to include variable neighborhood structure as used in the VNS heuristic.

REFERENCES

1. Moreno, J., *Who shall survive?: A new approach to the problem of human interrelations.* 1934.
2. Barnes, J., *Class and committees in a Norwegian island parish.* Human relations, 1954. **7**(1): p. 39.
3. Scott, J., *Social Network Analysis.* Sociology, 1988. **22**(1): p. 109-127.
4. Boginski, V., S. Butenko, and P. Pardalos, *On Structural Properties of the Market Graph In: Inovation in Financial and Economic Networks, Nagurney, A (ed.).* 2003: p. 28-45.
5. Boginski, V., S. Butenko, and P.M. Pardalos, *Mining market data: A network approach.* Computers & Operations Research, 2006. **33**(11): p. 3171-3184.
6. Butenko, S., *Maximum independent set and related problems, with applications*. 2003, Ph.D. Dissertation, University of Florida.
7. Mantegna, R. and H. Stanley, *An introduction to econophysics: correlations and complexity in finance*. 2000: Cambridge Univ Press, Cambridge, UK.
8. Tse, C., J. Liu, and F. Lau, *A network perspective of the stock market.* Journal of Empirical Finance, 2010. **17**(4): p. 659-667.
9. Balasundaram, B., et al., *Clique relaxations in social network analysis: The maximum k-plex problem*. 2008.
10. Mokken, R., *Cliques, clubs and clans.* Quality and Quantity, 1979. **13**(2): p. 161-173.
11. Alba, R., *A graph-theoretic definition of a sociometric clique.* The Journal of Mathematical Sociology, 1973. **3**(1): p. 113-126.
12. Luce, R., *Connectivity and generalized cliques in sociometric group structure.* Psychometrika, 1950. **15**(2): p. 169-190.
13. Seidman, S. and B. Foster, *A graph-theoretic generalization of the clique concept.* The Journal of Mathematical Sociology, 1978. **6**(1): p. 139-154.
14. Balasundaram, B., *Graph theoretic generalizations of clique: Optimization and extensions*. 2007, Ph.D. Dissertation, Texas A&M University.
15. Garey, M. and D. Johnson, *A Guide to the Theory of NP-Completeness.* Computers and Intractability, 1979. **3**(5): p. 23-26.
16. Harary, F. and I. Ross, *A procedure for clique detection using the group matrix.* Sociometry, 1957. **20**(3): p. 205-215.
17. Bomze, I.M., et al., *The maximum clique problem.* Handbook of combinatorial optimization, 1999. **4**(1): p. 1-74.

18. Tarjan, R. and A. Trojanowski, *Finding a maximum independent set*. 1976: Computer Science Department, Stanford University.
19. Robson, J., *Algorithms for maximum independent sets.* Journal of Algorithms, 1986. **7**(3): p. 425-440.
20. Dessmark, A., K. Jansen, and A. Lingas, *The maximum k-dependent and f-dependent set problem.* Algorithms and Computation, 1993: p. 88-97.
21. Djidev, H., et al., *On the Maximum k-depedent set problem, Technical Report*. 1992: Department of Computer Science, Lund University, Sweden. p. 92-91.
22. Trukhanov, S., *Novel approaches for solving large-scale optimization problems on graphs*. 2008, Ph.D. Dissertation, Texas A&M University.
23. Moser, H., R. Niedermeier, and M. Sorge, *Algorithms and experiments for clique relaxations—finding maximum s-plexes.* Experimental Algorithms, 2009: p. 233-244.
24. Johnson, D. and M. Trick, *Cliques, Coloring, and Satisfiability. Second DIMACS Implementation Challenge, volume 26 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, 1996.
25. Dimacs, *Cliques, Coloring, and Satisfiability: Second Dimacs Implementation Challenge.* 1995. Online: http://dimacs.rutgers.edu/Challenges/.
26. Kirkpatrick, S., *Optimization by simulated annealing: Quantitative studies.* Journal of Statistical Physics, 1984. **34**(5): p. 975-986.
27. Glover, F. and M. Laguna, *Tabu search*. 1997, Kluwer Academic Publishers, Boston.
28. Glover, F., *Tabu Search: A Tutorial.* Interfaces, 1990. **20**(4): p. 74-94.
29. Glover, F., *Tabu search-part II.* ORSA journal on Computing, 1990. **2**(1): p. 4-32.
30. Feo, T.A. and M.G.C. Resende, *A probabilistic heuristic for a computationally difficult set covering problem* 1.* Operations research letters, 1989. **8**(2): p. 67-71.
31. Feo, T.A. and M.G.C. Resende, *Greedy randomized adaptive search procedures.* Journal of Global Optimization, 1995. **6**(2): p. 109-133.
32. Kiehn, E. and J. Holland, *Membrane and nonmembrane proteins of mammalian cells. Organ, species, and tumor specificities.* Biochemistry, 1970. **9**(8): p. 1729-1738.
33. Goldberg, D., *Genetic algorithms in search and optimization*. 1989, Reading, MA: Addison-Wesley.
34. Hansen, P. and N. Mladenovic, *An introduction to VNS.* Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, 1998.
35. Festa, P., *Greedy randomized adaptive search procedures.* AIROnews, 2003. **7**(4): p. 7-11.
36. Festa, P. and M. Resende, *GRASP: basic components and enhancements.* Telecommunication Systems, 2010: p. 1-19.
37. Festa, P. and M.G.C. Resende, *GRASP: An annotated bibliography.* Essays and surveys in metaheuristics, 2002: p. 325–367.
38. Festa, P. and M.G.C. Resende, *An annotated bibliography of GRASP–Part II: Applications.* International Transactions in Operational Research, 2009. **16**(2): p. 131-172.

39. Feo, T.A., M.G.C. Resende, and S.H. Smith, *A Greedy Randomized Adaptive Search Procedure for Maximum Indepedent Set.* Operations Research, 1994. **42**(5): p. 860-878.
40. Hasselberg, J., P. Pardalos, and G. Vairaktarakis, *Test case generators and computational results for the maximum clique problem.* Journal of Global Optimization, 1993. **3**(4): p. 463-482.
41. Pardalos, P. and J. Xue, *The maximum clique problem.* Journal of Global Optimization, 1994. **4**(3): p. 301-328.
42. Abello, J., P. Pardalos, and M. Resende, *On maximum clique problems in very large graphs.* External memory algorithms, 1999. **50**: p. 119–130.
43. Resende, M.G.C., *Metaheuristic Hybridization with GRASP*. Tutorials in Operations Research. Inst. for Mgmt Sci. and OR, 2008.

APPPENDIX A

RESULTS OF

GRASP

V/S

BC EXACT ALGORITHM

FOR THE MAXIMUM CO-K-PLEX AND WEIGHTED CO-K-PLEX PROBLEMS

Appendix A presents comparison between GRASP and BC algorithms for MCPP-k and MWCPP-k. Table 11 shows performance of the GRASP algorithm for 10, 100 and 1000 iterations. Table 12, 13 and 14 show effectiveness of GRASP algorithm by comparing with results obtained by BC algorithm.

Graph\No. of Iterations	co-1-plex size (A(G))			Time (secs)		
	10	100	1000	10	100	1000
brock200_1.clq	20	20	21	< 0.001	0.047	0.468
brock200_2.clq	10	10	12	< 0.001	0.047	0.312
brock200_4.clq	15	16	16	< 0.001	0.047	0.359
brock400_2.clq	24	24	24	0.016	0.125	1.187
brock400_4.clq	25	25	25	0.031	0.125	1.187
brock800_2.clq	19	19	20	0.031	0.328	3.219
brock800_4.clq	20	20	20	0.047	0.343	3.14
c-fat200-1.clq	12	12	12	< 0.001	0.063	0.406
c-fat200-2.clq	24	24	24	< 0.001	0.078	0.688
c-fat200-5.clq	58	58	58	0.031	0.359	3.359
c-fat500-1.clq	14	14	14	0.015	0.218	2.047
c-fat500-2.clq	26	26	26	0.031	0.265	2.531
c-fat500-5.clq	64	64	64	0.078	0.719	7.016
c-fat500-10.clq	126	126	126	0.328	3.188	31.141
hamming6-2.clq	32	32	32	< 0.001	0.062	0.344
hamming6-4.clq	4	4	4	< 0.001	0.032	0.047
hamming8-2.clq	128	128	128	0.14	1.422	13.876
hamming8-4.clq	16	16	16	< 0.001	0.047	0.438
hamming10-2.clq	512	512	512	6.938	72.233	716.739
hamming10-4.clq	40	40	40	0.078	0.625	6.187

Table 11. GRASP results for 10, 100, and 1000 iterations for the MCPP-1

Graph\No. of Iterations	co-1-plex size (A(G))			Time (secs)		
	10	100	1000	10	100	1000
johnson8-2-4.clq	4	4	4	< 0.001	0.015	0.016
johnson8-4-4.clq	14	14	14	< 0.001	0.032	0.109
keller4.clq	11	11	11	< 0.001	0.031	0.234
keller5.clq	25	27	27	0.031	0.313	3.031
MANN_a9.clq	16	16	16	< 0.001	0.015	0.109
MANN_a27.clq	126	126	126	0.235	2.171	21.172
MANN_a45.clq	343	343	343	4.25	40.687	401.737
p_hat300-1.clq	8	8	8	0.015	0.078	0.719
p_hat300-2.clq	25	25	25	0.015	0.125	1.093
p_hat300-3.clq	35	36	36	0.031	0.156	1.484
p_hat700-1.clq	10	10	11	0.047	0.36	3.562
p_hat700-2.clq	44	44	44	0.078	0.61	5.938
p_hat700-3.clq	62	62	62	0.11	0.875	8.657

Table 11. GRASP results for 10, 100, and 1000 iterations for MCPP-1 (continued)

Graphs	BC						GRASP					
	$\alpha(G)$	$\alpha_2(G)$	$\alpha_3(G)$	Time (secs)			A(G)	$A_2(G)$	$A_3(G)$	Time (secs)		
brock200_1.clq	[20,31]	[25, 53]	[29, 70]	> 10800	> 10800	> 10800	20	25	28	< 0.001	< 0.001	0.015
brock200_2.clq	12	[13, 24]	[15, 37]	152.5	> 10800	> 10800	10	13	14	< 0.001	0.016	< 0.001
brock200_4.clq	17	[19, 41]	[22, 58]	6617.5	> 10800	> 10800	15	19	21	< 0.001	0.016	0.015
brock400_2.clq	[24, 68]	[27, 133]	[32, 151]	> 10800	> 10800	> 10800	24	28	32	0.016	0.015	0.031
brock400_4.clq	[23, 69]	[27, 133]	[32, 151]	> 10800	> 10800	> 10800	25	27	31	0.031	0.015	0.031
brock800_2.clq	[19, 116]	[23, 253]	[26, 305]	> 10800	> 10800	> 10800	19	22	25	0.031	0.047	0.047
brock800_4.clq	[19, 108]	[23, 252]	[26, 304]	> 10800	> 10800	> 10800	20	22	25	0.047	0.046	0.063
c-fat200-1.clq	12	12	12	17.1	148.9	6	12	12	10	< 0.001	< 0.001	< 0.001
c-fat200-2.clq	24	24	24	10.4	19.1	2.828	24	24	16	< 0.001	< 0.001	< 0.001
c-fat200-5.clq	58	58	58	2.1	2.1	1.125	58	58	33	0.031	0.031	0.015
c-fat500-1.clq	14	14	14	1334.4	1356.1	39.734	14	14	11	0.015	0.015	0.015
c-fat500-2.clq	26	26	26	535.7	605.3	33.437	26	26	17	0.031	0.015	0.016
c-fat500-5.clq	64	64	64	141.6	141.5	45.86	64	64	36	0.078	0.062	0.031
c-fat500-10.clq	126	126	126	39.3	76.5	13.547	126	126	67	0.328	0.172	0.063
hamming6-2.clq	32	32	32	< 0.001	< 0.001	96.094	32	32	32	< 0.001	< 0.001	< 0.001
hamming6-4.clq	4	6	8	0.2	0.3	0.407	4	6	8	< 0.001	< 0.001	< 0.001
hamming8-2.clq	128	128	[128, 143]	< 0.001	189.5	> 10800	128	128	128	0.14	0.156	0.203
hamming8-4.clq	16	16	[20, 32]	52.2	8115.2	> 10800	16	16	20	< 0.001	< 0.001	0.015
hamming10-2.clq	512	[512, 530]	[512, 566]	0.8	> 10800	> 10800	512	512	512	6.938	8.844	15.656
hamming10-4.clq	[36, 234]	[41, 153]	[51, 286]	> 10800	> 10800	> 10800	40	44	53	0.078	0.109	0.172

Table 12. Comparison of running time and solution quality between GRASP and Balasundaram's BC algorithm for the MCPP-k, k =1, 2, 3

Graphs	BC						GRASP					
	$\alpha(G)$	$\alpha_2(G)$	$\alpha_3(G)$	Time (secs)			$A(G)$	$A_2(G)$	$A_3(G)$	Time (secs)		
johnson8-2-4.clq	4	5	8	< 0.001	< 0.001	0.047	4	5	8	< 0.001	< 0.001	< 0.001
johnson8-4-4.clq	14	14	18	0.1	4.4	28.75	14	14	18	< 0.001	< 0.001	< 0.001
keller4.clq	11	15	21	129.8	365.4	737.781	11	15	21	< 0.001	< 0.001	< 0.001
keller5.clq	[26, 50]	[31, 83]	[40, 167]	> 10800	> 10800	> 10800	25	31	39	0.031	0.046	0.094
MANN_a9.clq	16	26	36	< 0.001	< 0.001	0.015	16	26	36	< 0.001	< 0.001	< 0.001
MANN_a27.clq	126	236	351	430.3	79.8	0.031	126	236	351	0.235	0.828	1.5
MANN_a45.clq	[344, 347]	[662, 668]	990	> 10800	> 10800	0.141	343	662	990	4.25	16.343	33.718
p_hat300-1.clq	8	[9, 66]	[12, 29]	127	> 10800	> 10800	8	10	12	0.015	0.015	0.016
p_hat300-2.clq	[25, 51]	[28, 85]	[36, 78]	> 10800	> 10800	> 10800	25	30	36	0.015	0.047	0.031
p_hat300-3.clq	[35, 71]	[43, 108]	[50, 112]	> 10800	> 10800	> 10800	35	43	51	0.031	0.031	0.046
p_hat700-1.clq	[11, 40]	[10, 291]	[13, 296]	> 10800	> 10800	> 10800	10	11	13	0.047	0.046	0.046
p_hat700-2.clq	[44, 208]	[50, 298]	[58, 298]	> 10800	> 10800	> 10800	44	52	60	0.078	0.125	0.141
p_hat700-3.clq	[62, 201]	[73, 311]	[83, 293]	> 10800	> 10800	> 10800	62	75	87	0.11	0.187	0.282

Table 12. Comparison of running time and solution quality between GRASP and Balasundaram's BC algorithm for the MCPP-k, k =1, 2, 3 (continued)

Graph	BC			GRASP		
	$\alpha_1^W(G)$	$\alpha_2^W(G)$	$\alpha_3^W(G)$	$\alpha_1^W(G)$	$\alpha_2^W(G)$	$\alpha_3^W(G)$
brock200_1.clq	28455	[34503, 62211]	[39440, 69381]	28455	28455	37683
brock200_2.clq	14556	17536	[20564, 20566]	14556	17334	19887
brock200_4.clq	21324	25889	[31326, 55398]	21324	21324	28717
brock400_2.clq	[67233, 225025]	[75217, 272498]	[93337, 307543]	63392	75411	88076
brock400_4.clq	[67102, 210135]	[79529, 272183]	[95056, 337542]	67102	76421	88096
brock800_2.clq	[111150, 916516]	[127835, 1376970]	[143897, 1375300]	109374	128849	141235
brock800_4.clq	[109273, 918006]	[140704, 1381870]	[163373, 1365850]	110473	132771	151756
c-fat200-1.clq	15420	15420	15882	15420	15420	15013
c-fat200-2.clq	27112	27112	27112	27112	27112	21301
c-fat200-5.clq	65414	65414	65414	65414	65414	65414
c-fat500-1.clq	45187	45187	45187	45187	45187	32716
c-fat500-2.clq	76178	76178	76178	76178	76178	55507
c-fat500-5.clq	179508	179508	179508	179508	179508	148842
c-fat500-10.clq	338046	338046	338046	338046	338046	338046
hamming6-2.clq	11635	11635	12100	11635	11635	12100
hamming6-4.clq	2021	2896	3808	2021	2894	3547
hamming8-2.clq	169034	175098	[187101, 196159]	169034	169034	188359
hamming8-4.clq	28799	[32619, 124196]	[40595, 93721]	28799	28799	37518
hamming10-2.clq	[2618690, 2618780]	[2650000, 2790000]	[2684660, 3067230]	2618690	2665520	2475380
hamming10-4.clq	2618690	[353752, 2433450]	[451272, 2429940]	277650	277650	417314

Table 13. Comparison of solution quality between GRASP and Balasundaram's BC algorithm for the MWCPP-k, k =1, 2, 3

Graph	BC			GRASP		
	$\alpha_1^W(G)$	$\alpha_2^W(G)$	$\alpha_3^W(G)$	$\alpha_1^W(G)$	$\alpha_2^W(G)$	$\alpha_3^W(G)$
johnson8-2-4.clq	925	1216	1731	925	1216	1731
johnson8-4-4.clq	7508	7685	9824	7508	7672	9824
keller4.clq	14140	20058	[24411, 24413]	13352	20058	24015
keller5.clq	[140699, 506572]	[187274, 1254230]	[241092, 1274340]	128431	170354	206480
MANN_a9.clq	5596	7362	9111	5596	7362	9111
MANN_a27.clq	[299319, 405511]	577927	700216	343159	343159	700216
MANN_a45.clq	[2573130, 2573150]	4301300	5203950	2571200	4301300	5203950
p_hat300-1.clq	16303	20455	24899	16303	16303	22727
p_hat300-2.clq	[40355, 40359]	[50890, 101732]	[59927, 130841]	40355	50028	58458
p_hat300-3.clq	[65947, 162690]	[78285, 161837]	[93069, 178645]	65023	77059	87147
p_hat700-1.clq	[50462, 648095]	[54871, 951804]	[67901, 1110503]	47037	47037	70408
p_hat700-2.clq	[173232, 683481]	[215491, 1045690]	[252983, 1081170]	174048	212077	244995
p_hat700-3.clq	[254492, 721752]	[295874, 1075250]	[349467, 1085640]	251397	294538	334295

Table 13. Comparison of solution quality between GRASP and Balasundaram's BC algorithm for the MWCPP-k, k =1, 2, 3 (continued)

Graph	BC			GRASP		
	Time (secs)					
	k=1	k=2	k=3	k=1	k=2	k=3
brock200_1.clq	33.813	> 3600	> 3600	0.016	0.016	0.016
brock200_2.clq	8.141	328.296	1175.45	< 0.001	0.016	< 0.001
brock200_4.clq	37.781	2985.45	> 3600	< 0.001	< 0.001	0.016
brock400_2.clq	> 3600	> 3600	> 3600	0.015	0.031	0.031
brock400_4.clq	> 3600	> 3600	> 3600	0.016	0.031	0.031
brock800_2.clq	> 3600	> 3600	> 3600	0.032	0.031	0.047
brock800_4.clq	> 3600	> 3600	> 3600	0.031	0.031	0.047
c-fat200-1.clq	4.672	8.078	6.328	< 0.001	< 0.001	0.015
c-fat200-2.clq	5.157	12.235	17.25	< 0.001	0.015	0.016
c-fat200-5.clq	2.515	5.687	6.968	0.031	0.016	0.031
c-fat500-1.clq	271.89	153.031	2091.22	< 0.001	< 0.001	0.015
c-fat500-2.clq	288.109	275.453	405.047	0.015	0.015	0.016
c-fat500-5.clq	271.938	247.765	288	0.046	0.046	0.031
c-fat500-10.clq	112.235	129.422	190.39	0.234	0.234	0.078
hamming6-2.clq	0.109	0.078	0.563	< 0.001	< 0.001	0.015
hamming6-4.clq	0.25	1.031	0.625	< 0.001	< 0.001	< 0.001
hamming8-2.clq	0.204	8.625	> 3600	0.141	0.141	0.641
hamming8-4.clq	2.985	> 3600	> 3600	< 0.001	< 0.001	0.015
hamming10-2.clq	3.984	> 3600	> 3600	9.203	39.703	57.265
hamming10-4.clq	3.984	> 3600	> 3600	0.078	0.078	0.282

Table 14. Comparison of running time between GRASP and Balasundaram’s BC algorithm for the MWCPP-k, k =1, 2, 3

Graph	BC			GRASP		
	Time (secs)					
	k=1	K =2	k=3	k=1	k=2	k=3
johnson8-2-4.clq	0.171	0.188	0.125	< 0.001	< 0.001	< 0.001
johnson8-4-4.clq	0.14	4.797	8.266	< 0.001	< 0.001	0.016
keller4.clq	1.266	503	750.266	0.015	< 0.001	0.016
keller5.clq	> 3600	> 3600	≥ 3600	0.031	0.062	0.11
MANN_a9.clq	0.094	0.062	0.078	< 0.001	< 0.001	< 0.001
MANN_a27.clq	> 3600	0.156	0.125	0.421	0.421	0.922
MANN_a45.clq	0.344	0.328	0.203	10.437	36.625	23.109
p_hat300-1.clq	28.204	417.921	1043.4	< 0.001	< 0.001	0.015
p_hat300-2.clq	112.547	> 3600	> 3600	< 0.001	0.032	0.031
p_hat300-3.clq	755.641	> 3600	> 3600	0.031	0.047	0.078
p_hat700-1.clq	> 3600	> 3600	> 3600	0.015	0.015	0.016
p_hat700-2.clq	> 3600	> 3600	> 3600	0.063	0.141	0.203
p_hat700-3.clq	> 3600	> 3600	> 3600	0.141	0.297	0.516

Table 14. Comparison of running time between GRASP and Balasundaram’s BC algorithm for the MWCPP-k, k =1, 2, 3 (continued)

APPPENDIX B

RESULTS OF

GRASP ALGORITHM FOR MCPP-k AND MWCPP-k

Appendix B presents overall GRASP results on both combinatorial optimization problems, MCPP-k and MWCPP-k. Tables 15, 16, and 17 present the statistics collected on GRASP algorithm for the maximum co-k-plex problem. Also, Tables 18, 19 and 20 present the statistics collected on GRASP algorithm for the maximum weighted co-k-plex problem.

Graph	A(G)	CTime (secs)	LSTime (secs)	GRASPTime (secs)	LSHitRate	LSAvgPerInc
brock200_1.clq	20	< 0.001	< 0.001	< 0.001	0.7	0.116
brock200_2.clq	10	< 0.001	< 0.001	< 0.001	0.2	0.292
brock200_4.clq	15	< 0.001	< 0.001	< 0.001	0.2	0.084
brock400_2.clq	24	< 0.001	< 0.001	0.016	0.5	0.126
brock400_4.clq	25	0.016	< 0.001	0.031	0.6	0.130
brock800_2.clq	19	0.031	< 0.001	0.031	0.4	0.116
brock800_4.clq	20	0.031	0.016	0.047	0.5	0.141
c-fat200-1.clq	12	< 0.001	< 0.001	< 0.001	0	0
c-fat200-2.clq	24	< 0.001	< 0.001	< 0.001	0	0
c-fat200-5.clq	58	0.016	0.015	0.031	0	0
c-fat500-1.clq	14	0.015	< 0.001	0.015	0	0
c-fat500-2.clq	26	< 0.001	0.031	0.031	0	0
c-fat500-5.clq	64	0.078	< 0.001	0.078	0	0
c-fat500-10.clq	126	0.281	0.016	0.328	0	0
hamming6-2.clq	32	< 0.001	< 0.001	< 0.001	0	0
hamming6-4.clq	4	< 0.001	< 0.001	< 0.001	0	0
hamming8-2.clq	128	0.11	0.015	0.14	0	0
hamming8-4.clq	16	< 0.001	< 0.001	< 0.001	0.4	0.260
hamming10-2.clq	512	6.594	0.281	6.938	0	0.000
hamming10-4.clq	40	0.047	0.015	0.078	0.9	0.201

Table 15. Results of GRASP for the maximum co-1-plex problem on DIMACS instances

Graph	A(G)	CTime (secs)	LSTime (secs)	GRASPTime (secs)	LSHitRate	LSAvgPerInc
johnson8-2-4.clq	4	< 0.001	< 0.001	< 0.001	0	0
johnson8-4-4.clq	14	< 0.001	< 0.001	< 0.001	0.3	0.409
keller4.clq	11	< 0.001	< 0.001	< 0.001	0.6	0.151
keller5.clq	25	0.031	< 0.001	0.031	0.8	0.157
MANN_a9.clq	16	< 0.001	< 0.001	< 0.001	0.1	0.067
MANN_a27.clq	126	0.203	0.016	0.235	0.3	0.008
MANN_a45.clq	343	3.984	0.219	4.25	0	0
p_hat300-1.clq	8	< 0.001	< 0.001	0.015	0.2	0.155
p_hat300-2.clq	25	< 0.001	0.015	0.015	0.7	0.229
p_hat300-3.clq	35	< 0.001	0.015	0.031	0.9	0.091
p_hat700-1.clq	10	0.031	0.016	0.047	0.4	0.234
p_hat700-2.clq	44	0.015	0.047	0.078	0.9	0.428
p_hat700-3.clq	62	0.047	0.047	0.11	0.8	0.231

Table 15. Results of GRASP for the maximum co-1-plex problem on DIMACS instances (continued)

Graph	$A_2(G)$	CTime (secs)	LSTime (secs)	GRASPTime (secs)	LSHitRate	LSAvgPerInc
brock200_1.clq	25	< 0.001	< 0.001	< 0.001	0.6	0.127
brock200_2.clq	13	0.016	< 0.001	0.016	0.3	0.279
brock200_4.clq	19	< 0.001	< 0.001	0.016	0.4	0.259
brock400_2.clq	28	0.015	< 0.001	0.015	0.7	0.101
brock400_4.clq	27	0.015	< 0.001	0.015	0.8	0.128
brock800_2.clq	22	0.032	< 0.001	0.047	0.8	0.155
brock800_4.clq	22	0.015	0.031	0.046	0.8	0.085
c-fat200-1.clq	12	< 0.001	< 0.001	< 0.001	0.7	3.000
c-fat200-2.clq	24	< 0.001	< 0.001	< 0.001	0.7	3.582
c-fat200-5.clq	58	0.016	< 0.001	0.031	0.8	11.044
c-fat500-1.clq	14	0.015	< 0.001	0.015	0.2	6.000
c-fat500-2.clq	26	0.015	< 0.001	0.015	0.3	8.244
c-fat500-5.clq	64	0.047	0.015	0.062	1	12.832
c-fat500-10.clq	126	0.047	0.109	0.172	1	25.313
hamming6-2.clq	32	< 0.001	< 0.001	< 0.001	0.1	0.333
hamming6-4.clq	6	< 0.001	< 0.001	< 0.001	0.4	0.500
hamming8-2.clq	128	0.109	0.032	0.156	0.1	0.096
hamming8-4.clq	16	< 0.001	< 0.001	< 0.001	0.8	0.367
hamming10-2.clq	512	7.314	1.467	8.844	0.1	0.183
hamming10-4.clq	44	0.015	0.079	0.109	1	0.350

Table 16. Results of GRASP for the maximum co-2-plex problem on DIMACS instances

Graph	$A_2(G)$	CTime (secs)	LSTime (secs)	GRASPTime (secs)	LSHitRate	LSAvgPerInc
johnson8-2-4.clq	5	< 0.001	< 0.001	< 0.001	0	0
johnson8-4-4.clq	14	< 0.001	< 0.001	< 0.001	0.2	0.220
keller4.clq	15	< 0.001	< 0.001	< 0.001	0.6	0.091
keller5.clq	31	0.031	0.015	0.046	0.7	0.176
MANN_a9.clq	26	< 0.001	< 0.001	< 0.001	0.4	0.053
MANN_a27.clq	236	0.733	0.079	0.828	0.1	0.009
MANN_a45.clq	662	15.749	0.485	16.343	0	0
p_hat300-1.clq	10	0.015	< 0.001	0.015	0.4	0.280
p_hat300-2.clq	30	0.016	< 0.001	0.047	0.9	0.313
p_hat300-3.clq	43	0.031	< 0.001	0.031	0.8	0.117
p_hat700-1.clq	11	0.031	< 0.001	0.046	0.7	0.197
p_hat700-2.clq	52	0.062	0.048	0.125	0.9	0.374
p_hat700-3.clq	75	0.094	0.078	0.187	1	0.342

Table 16. Results of GRASP for the maximum co-2-plex problem on DIMACS instances (continued)

Graph	$A_3(G)$	CTime (secs)	LSTime (secs)	GRASPTime (secs)	LSHitRate	LSAvgPerInc
brock200_1.clq	28	< 0.001	< 0.001	0.015	0.8	0.196
brock200_2.clq	14	< 0.001	< 0.001	< 0.001	0.7	0.238
brock200_4.clq	21	< 0.001	< 0.001	0.015	0.7	0.235
brock400_2.clq	32	0.015	0.016	0.031	0.9	0.111
brock400_4.clq	31	0.015	< 0.001	0.031	0.8	0.148
brock800_2.clq	25	0.047	< 0.001	0.047	1	0.127
brock800_4.clq	25	0.032	0.015	0.063	1	0.136
c-fat200-1.clq	10	< 0.001	< 0.001	< 0.001	0.3	0.333
c-fat200-2.clq	16	< 0.001	< 0.001	< 0.001	0.3	0.333
c-fat200-5.clq	33	< 0.001	0.015	0.015	0.5	8.350
c-fat500-1.clq	11	0.015	< 0.001	0.015	0.2	0.333
c-fat500-2.clq	17	0.016	< 0.001	0.016	0.3	1.306
c-fat500-5.clq	36	0.031	< 0.001	0.031	0.2	4.167
c-fat500-10.clq	67	0.016	0.031	0.063	0.3	10.611
hamming6-2.clq	32	< 0.001	< 0.001	< 0.001	0.6	0.173
hamming6-4.clq	8	< 0.001	< 0.001	< 0.001	0.3	0.422
hamming8-2.clq	128	0.109	0.078	0.203	0.6	0.090
hamming8-4.clq	20	< 0.001	< 0.001	0.015	0.9	0.284
hamming10-2.clq	512	7.017	8.592	15.656	0.6	0.080
hamming10-4.clq	53	0.093	0.063	0.172	1	0.246

Table 17. Results of GRASP for the maximum co-3-plex problem on DIMACS instances

Graph	$A_3(G)$	CTime (secs)	LSTime (secs)	GRASPTime (secs)	LSHitRate	LSAvgPerInc
johnson8-2-4.clq	8	< 0.001	< 0.001	< 0.001	1	0.276
johnson8-4-4.clq	18	< 0.001	< 0.001	< 0.001	0.9	0.212
keller4.clq	21	< 0.001	< 0.001	< 0.001	1	0.159
keller5.clq	39	0.016	0.062	0.094	0.9	0.132
MANN_a9.clq	36	< 0.001	< 0.001	< 0.001	0.4	0.160
MANN_a27.clq	351	1.375	0.094	1.5	0.4	0.137
MANN_a45.clq	990	33.049	0.576	33.718	0.1	0.092
p_hat300-1.clq	12	< 0.001	< 0.001	0.016	0.7	0.304
p_hat300-2.clq	36	0.015	< 0.001	0.031	0.6	0.272
p_hat300-3.clq	51	< 0.001	0.046	0.046	0.9	0.317
p_hat700-1.clq	13	0.046	< 0.001	0.046	0.9	0.292
p_hat700-2.clq	60	0.047	0.078	0.141	0.9	0.504
p_hat700-3.clq	56	0.062	0.063	0.141	1	0.345

Table 17. Results of GRASP for the maximum co-3-plex problem on DIMACS instances (continued)

Graph	I(G)	α_1^W (G)	CTime (secs)	LSTime (secs)	GRASPTime (secs)	LSHitRate	LSAvgPerInc
brock200_1.clq	18	28455	< 0.001	< 0.001	0.016	0.9	0.253
brock200_2.clq	9	14556	< 0.001	< 0.001	< 0.001	1	0.307
brock200_4.clq	13	21324	< 0.001	< 0.001	< 0.001	0.9	0.268
brock400_2.clq	21	63392	< 0.001	0.015	0.015	1	0.174
brock400_4.clq	21	67102	< 0.001	< 0.001	0.016	0.8	0.192
brock800_2.clq	18	109374	< 0.001	0.016	0.032	0.9	0.262
brock800_4.clq	17	110473	0.015	< 0.001	0.031	0.9	0.280
c-fat200-1.clq	12	15420	< 0.001	< 0.001	< 0.001	0	0
c-fat200-2.clq	22	27112	< 0.001	< 0.001	< 0.001	0	0
c-fat200-5.clq	58	65414	< 0.001	0.015	0.031	0	0
c-fat500-1.clq	12	45187	< 0.001	< 0.001	< 0.001	0	0
c-fat500-2.clq	26	76178	< 0.001	< 0.001	0.015	0	0
c-fat500-5.clq	64	179508	0.046	< 0.001	0.046	0	0
c-fat500-10.clq	125	338046	0.202	0.016	0.234	0	0
hamming6-2.clq	32	11635	< 0.001	< 0.001	< 0.001	0.8	0.373
hamming6-4.clq	4	2021	< 0.001	< 0.001	< 0.001	0.2	0.129
hamming8-2.clq	128	169034	0.032	0.093	0.141	1	0.374
hamming8-4.clq	16	28799	< 0.001	< 0.001	< 0.001	0.9	0.449
hamming10-2.clq	512	2618690	1.328	7.812	9.203	1	0.507
hamming10-4.clq	34	277650	0.031	0.031	0.078	1	0.372

Table 18. Results of GRASP for the maximum weighted co-1-plex problem on DIMACS instances

Graph	I(G)	α_1^W (G)	CTime (secs)	LSTime (secs)	GRASPTime (secs)	LSHitRate	LSAvgPerInc
johnson8-2-4.clq	4	925	< 0.001	< 0.001	< 0.001	0	0
johnson8-4-4.clq	14	7508	< 0.001	< 0.001	< 0.001	0.9	0.257
keller4.clq	11	13352	< 0.001	< 0.001	0.015	0.7	0.214
keller5.clq	20	128431	< 0.001	0.016	0.031	1	0.255
MANN_a9.clq	16	5596	< 0.001	< 0.001	< 0.001	0.8	0.154
MANN_a27.clq	119	343159	0.155	0.251	0.421	0.9	0.072
MANN_a45.clq	331	2571200	3.657	6.733	10.437	1	0.079
p_hat300-1.clq	7	16303	< 0.001	< 0.001	< 0.001	0.6	0.312
p_hat300-2.clq	23	40355	< 0.001	< 0.001	< 0.001	1	0.471
p_hat300-3.clq	32	65023	< 0.001	0.016	0.031	1	0.250
p_hat700-1.clq	9	47037	< 0.001	< 0.001	0.015	0.9	0.403
p_hat700-2.clq	40	174048	< 0.001	0.047	0.063	1	0.545
p_hat700-3.clq	59	251397	0.015	0.11	0.141	1	0.668

Table 18. Results of GRASP for the maximum weighted co-1-plex problem on DIMACS instances (continued)

Graph	I_2(G)	α_2^W (G)	CTime (secs)	LSTime (secs)	GRASPTime (secs)	LSHitRate	LSAvgPerInc
brock200_1.clq	22	34503	< 0.001	0.015	0.031	1	0.159
brock200_2.clq	11	17334	< 0.001	< 0.001	0.016	0.8	0.181
brock200_4.clq	16	24239	< 0.001	< 0.001	0.016	0.8	0.202
brock400_2.clq	25	75411	< 0.001	0.015	0.031	1	0.119
brock400_4.clq	25	76421	< 0.001	0.015	0.031	0.9	0.313
brock800_2.clq	21	128849	< 0.001	0.016	0.031	1	0.223
brock800_4.clq	21	132771	< 0.001	0.015	0.031	0.8	0.411
c-fat200-1.clq	12	15420	< 0.001	< 0.001	< 0.001	0.5	3.860
c-fat200-2.clq	22	27112	< 0.001	< 0.001	0.015	0.7	5.492
c-fat200-5.clq	58	65414	< 0.001	0.016	0.016	1	8.747
c-fat500-1.clq	12	45187	< 0.001	< 0.001	< 0.001	0.6	5.179
c-fat500-2.clq	26	76178	< 0.001	< 0.001	< 0.001	0.5	3.444
c-fat500-5.clq	64	179508	< 0.001	0.015	0.015	0.7	18.799
c-fat500-10.clq	125	338046	0.032	0.077	0.125	0.9	23.407
hamming6-2.clq	32	11635	< 0.001	< 0.001	< 0.001	1	0.191
hamming6-4.clq	6	2894	< 0.001	< 0.001	< 0.001	0.1	0.318
hamming8-2.clq	128	175098	0.047	0.25	0.328	1	0.177
hamming8-4.clq	16	31099	< 0.001	0.016	0.016	0.9	0.411
hamming10-2.clq	512	2665520	2.155	37.485	39.703	1	0.270
hamming10-4.clq	43	338057	0.032	0.14	0.187	1	0.672

Table 19. Results of GRASP for the maximum weighted co-2-plex problem on DIMACS instances

Graph	I_2(G)	α_2^W (G)	CTime (secs)	LSTime (secs)	GRASPTime (secs)	LSHitRate	LSAvgPerInc
johnson8-2-4.clq	5	1216	< 0.001	< 0.001	< 0.001	0.6	0.181
johnson8-4-4.clq	14	7672	< 0.001	< 0.001	< 0.001	0.9	0.246
keller4.clq	14	20058	< 0.001	< 0.001	< 0.001	0.8	0.273
keller5.clq	27	170354	0.03	0.016	0.062	1	0.337
MANN_a9.clq	25	7362	< 0.001	< 0.001	< 0.001	1	0.129
MANN_a27.clq	234	577927	0.72	1.061	1.813	1	0.079
MANN_a45.clq	660	4301300	15.688	20.859	36.625	1	0.045
p_hat300-1.clq	9	20455	< 0.001	< 0.001	< 0.001	0.9	0.411
p_hat300-2.clq	28	50028	0.016	< 0.001	0.032	1	0.709
p_hat300-3.clq	41	77059	< 0.001	0.032	0.047	1	0.431
p_hat700-1.clq	11	61555	< 0.001	0.015	0.015	0.6	0.657
p_hat700-2.clq	47	212077	0.031	0.094	0.141	1	1.329
p_hat700-3.clq	67	294538	0.031	0.25	0.297	1	0.818

Table 19. Results of GRASP for the maximum weighted co-2-plex problem on DIMACS instances (continued)

Graph	I_3(G)	α_3^W(G)	CTime(secs)	LSTime(secs)	GRASPTime(secs)	LSHitRate	LSAvgPerInc
brock200_1.clq	25	37683	0.016	<0.001	0.016	0.9	0.115
brock200_2.clq	13	19887	<0.001	<0.001	<0.001	0.7	0.475
brock200_4.clq	19	28717	0.016	<0.001	0.016	0.8	0.340
brock400_2.clq	28	88076	<0.001	0.016	0.031	1	0.257
brock400_4.clq	30	88096	<0.001	0.015	0.031	1	0.309
brock800_2.clq	23	141235	0.016	0.015	0.047	1	0.323
brock800_4.clq	22	151756	0.015	0.016	0.047	1	0.205
c-fat200-1.clq	10	15013	<0.001	<0.001	0.015	0.7	1.299
c-fat200-2.clq	22	21301	<0.001	<0.001	0.016	0.3	1.117
c-fat200-5.clq	58	65414	<0.001	0.015	0.031	0.6	2.994
c-fat500-1.clq	10	32716	<0.001	<0.001	0.015	0.8	0.407
c-fat500-2.clq	17	55507	<0.001	<0.001	0.016	0.6	1.211
c-fat500-5.clq	62	148842	<0.001	0.016	0.031	0.6	1.060
c-fat500-10.clq	125	338046	0.047	0.016	0.078	0.3	0.069
hamming6-2.clq	28	12100	<0.001	<0.001	0.015	0.8	0.083
hamming6-4.clq	8	3547	<0.001	<0.001	<0.001	0.7	0.532
hamming8-2.clq	128	188359	0.142	0.483	0.641	1	0.216
hamming8-4.clq	18	37518	0.015	<0.001	0.015	1	0.600
hamming10-2.clq	366	2475380	3.628	53.59	57.265	1	0.154
hamming10-4.clq	51	417314	0.061	0.205	0.282	1	0.389

Table 20. Results of GRASP for the maximum weighted co-3-plex problem on DIMACS instances

Graph	I_3(G)	α_3^W (G)	CTime(secs)	LSTime(secs)	GRASPTime(secs)	LSHitRate	LSAvgPerInc
johnson8-2-4.clq	8	1731	< 0.001	< 0.001	< 0.001	0.3	0.050
johnson8-4-4.clq	18	9824	< 0.001	< 0.001	0.016	0.9	0.223
keller4.clq	18	24015	< 0.001	< 0.001	0.016	1	0.430
keller5.clq	38	206480	< 0.001	0.094	0.11	1	0.524
MANN_a9.clq	29	9111	< 0.001	< 0.001	< 0.001	0.6	0.101
MANN_a27.clq	351	700216	0.828	0.063	0.922	1	0.234
MANN_a45.clq	990	5203950	22.437	0.578	23.109	1	0.165
p_hat300-1.clq	10	22727	< 0.001	< 0.001	0.015	0.9	0.378
p_hat300-2.clq	31	58458	0.016	0.015	0.031	1	0.603
p_hat300-3.clq	42	87147	< 0.001	0.078	0.078	1	0.454
p_hat700-1.clq	13	70408	< 0.001	< 0.001	0.016	0.9	0.442
p_hat700-2.clq	57	244995	< 0.001	0.203	0.203	1	1.108
p_hat700-3.clq	82	334295	0.015	0.469	0.516	1	0.714

Table 20. Results of GRASP for the maximum weighted co-3-plex problem on DIMACS instances (continued)

VITA

Amol Atmaram Bhave

Candidate for the Degree of

Master of Science

Thesis: GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE FOR THE MAXIMUM CO-K-PLEX PROBLEM

Major Field: Industrial Engineering and Management

Biographical:

Education:

December, 2010
Master of Science in Industrial Engineering and Management
Oklahoma State University, Stillwater, OK, USA

June, 2006
Bachelor of Engineering in Mechanical Engineering
Shivaji University, Kolhapur, MH, India

Experience:

Research Assistant (January, 2009 - December, 2010)
Oklahoma State University, Stillwater, OK, USA

Research Engineer (September, 2007 – May, 2008)
Indian Institute of Technology, Bombay, Mumbai, MH, India

Junior Engineer (June, 2006 – August, 2007)
Atharva Engineering Company, Mumbai, MH, India

Professional Memberships:

Alpha Pi Mu
Institute for Operations Research and the Management Sciences
Society for Industrial and Applied Mathematics

Name: Amol Atmaram Bhave Date of Degree: December, 2010

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE FOR THE MAXIMUM CO-K-PLEX PROBLEM

Pages in Study: 57 Candidate for the Degree of Master of Science

Major Field: Industrial Engineering and Management

The focus of this thesis is a degree based relaxation of independent sets in graphs called co-k-plexes and the related combinatorial optimization problem of finding a maximum cardinality co-k-plex in G. This thesis develops a metaheuristic approach for solving the maximum co-k-plex problem which is known to be *NP-hard*. The approach is further extended for finding a maximum weighted co-k-plex in G where vertices of G are associated with specific weights. As the maximum co-k-plex problem in G is equivalent to the maximum k-plex problem in $\overline{G}$, many applications of this problem can be found in clustering and data mining social networks, biological networks, internet graphs and stock market graphs among others.

In this thesis, a Greedy Randomized Adaptive Search Procedure (GRASP) is developed to solve the maximum co-k-plex and maximum weighted co-k-plex problems. Computational experiments are performed to study the effectiveness of the proposed metaheuristic on benchmark instances. Finally, the performance of the developed GRASP algorithms for both versions was confirmed by comparing the running time and solution quality with results obtained by an exact algorithm.

ADVISER'S APPROVAL: Dr. Balabhaskar Balasundaram