2006

Simulation Of Random Set Covering Problems With Known Optimal Solutions And Explicitly Induced Correlations Amoong Coefficients

Nabin Sapkota
University of Central Florida

SIMULATION OF RANDOM SET COVERING PROBLEMS WITH KNOWN OPTIMAL SOLUTIONS AND EXPLICITLY INDUCED CORRELATION AMONG COEFFICIENTS

by

NABIN SAPKOTA
B.E. Regional Engineering College, Tamilnadu, India, 1998
M.S. University of Central Florida, 2003

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Industrial Engineering and Management Systems
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2006

Major Professor: Charles H. Reilly

ABSTRACT

The objective of this research is to devise a procedure to generate random Set Covering Problem (SCP) instances with known optimal solutions and correlated coefficients. The procedure presented in this work can generate a virtually unlimited number of SCP instances with known optimal solutions and realistic characteristics, thereby facilitating testing of the performance of SCP heuristics and algorithms.

A four-phase procedure based on the Karush-Kuhn-Tucker (KKT) conditions is proposed to generate SCP instances with known optimal solutions and correlated coefficients. Given randomly generated values for the objective function coefficients and the sum of the binary constraint coefficients for each variable and a randomly selected optimal solution, the procedure: (1) calculates the range for the number of possible constraints, (2) generates constraint coefficients for the variables with value one in the optimal solution, (3) assigns values to the dual variables, and (4) generates constraint coefficients for variables with value 0 in the optimal solution so that the KKT conditions are satisfied.

A computational demonstration of the procedure is provided. A total of 525 SCP instances are simulated under seven correlation levels and three levels for the number of constraints. Each of these instances is solved using three simple heuristic procedures. The

performance of the heuristics on the SCP instances generated is summarized and analyzed. The performance of the heuristics generally worsens as the expected correlation between the coefficients increases and as the number of constraints increases. The results provide strong evidence of the benefits of the procedure for generating SCP instances with correlated coefficients, and in particular SCP instances with known optimal solutions.

To my parents and friends who believed in me

ACKNOWLEDGMENTS

There were several people that impacted this work with their generous support. I am truly grateful to my advisor Dr. Charles H. Reilly for his constant support, motivation and encouragement throughout the research phase of the Ph.D. program. He was the principal source of inspiration while pursuing my dream of achieving the highest academic degree. His guidance and optimism gave me enough confidence to sail through the uncharted territory of research.

I would also like to thank my friend Mr. Sudhir Shakya for the time we spent together brainstorming about how to make the software codes foolproof and efficient. I would also like to thank my friend Mr. Ali Ahmad for his help in several occasions with his valuable advice regarding research techniques and software tools.

Last but not the least, I would like to thank my committee members for their productive advice, as well as, professors and staff members of the Department of Industrial Engineering and Management Systems at UCF for all the support they have extended over the years, making my experience at UCF pleasant and fruitful.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADH	Add/Drop Heuristic (for set covering problem)
AH	Add Heuristic (for set covering problem)
DH	Drop Heuristic (for set covering problem)
ECI	Explicit Correlation Induction
GAP	Generalized Assignment Problem
ICI	Implicit Correlation Induction
KKT	Karush-Kuhn-Tucker (conditions)
KP01	One Dimensional Knapsack Problem
SCP	Set Covering Problem
SVC	Superfluous Variable Condition
TSP	Traveling Salesman Problem

CHAPTER ONE: INTRODUCTION

The primary objective of this research is to devise a procedure for simulating random Set Covering Problem (SCP) instances with known optimal solutions and with specified population correlation among the coefficients. An additional objective is to develop a software program that will generate random, valid SCP instances given a specified optimal solution and correlated coefficients simulated based on a desired population correlation level.

This chapter begins with an overview of the SCP structure and its common applications. The chapter also explains the relevance of this research and its potential contributions to the field of optimization.

Definition of SCP

Let $\mathbf{A} = (a_{ij})$ be a binary $m \times n$ matrix and $\boldsymbol{c} = (c_j)$ be a positive integer-valued n-$n-$vector . Let the indices of the rows and columns of this matrix be represented by $I = \{1, 2, 3, ..., m\}$ and $J = \{1, 2, 3, ..., n\}$, respectively. The binary coefficients in each column of **A** represent a subset of I. Any column $j \in J$ covers row $i \in I$ if a_{ij}=1. The

cost of including the j^{th} subset in the solution (or cover) is c_j. The objective of the SCP is to choose a minimum-cost collection of subsets whose union covers *I*.

SCP may be formulated as follows. Define

$$x_j = \begin{cases} 1 \text{ if subset } j \text{ is included in the cover} \\ 0 \text{ otherwise} \end{cases}$$

for all $j \in J$. Then a complete mathematical representation of SCP is:

$$\text{Minimize } \sum_{j \in J} c_j x_j$$

Subject to the constraints $\sum_{j \in J} a_{ij} x_j \geq 1$ for all $i \in I$,

$x_j \in \{0,1\}$, for all $j \in J$,

where $a_{ij} \in \{0,1\} \ \forall \ i \in I, j \in J$.

The first constraint set includes m structural constraints that ensure that every row $i \in I$ is covered by at least one subset $j \in J$. The second constraint set requires that each subset is either included in the cover or not. The objective is to find a collection of columns (subsets) that covers all of the structural constraints at minimum total cost.

Applications of SCPs

There have been and are many practical applications of SCPs in various optimization scenarios. Balas and Padberg (1976) provide a bibliography on applications

of SCP. According to their paper, some of the diverse SCP application areas are crew scheduling (e.g., airlines and railroads), airline fleet scheduling, truck delivery, cutting stock, line and capacity balancing, facility location, capital investment, switching current design and symbolic logic, information retrieval, marketing and political districting. Other applications include bus crew scheduling (Smith, 1988), naval vessel scheduling (Brown, Graves, and Ronen, 1987; Fisher and Rosenwein, 1989), steel mill operations (Vasco, Wolf, and Stott, 1987), improving wireless sensor network lifetime (Cardei and Du, 2005), and preference scheduling for nurses (Bard and Purnomo, 2005). Certainly there have been many applications of SCPs in diverse settings over many years. Furthermore, there is every reason to think that SCP will continue to be an important optimization problem.

Relevance and Potential Contribution of the Research

New algorithms and heuristics to solve optimization problems, including SCP, are developed on an ongoing basis. Any new solution method should be tested for its efficacy and, if possible, comparative evaluations with other solution methods for the same class of optimization problems should be made. This practice would not only determine whether the newly coined solution procedure is trustworthy, but also would show how its performance compares to that of existing solution methods. In general, testing of solution procedures is necessary to determine their practical capabilities and limitations (Reilly,

1999).

Selection of test problem families and their pros and cons have been addressed by Reilly (1999). He explains that the inferences drawn from computational studies might be influenced by the family of problems chosen. An adequate number of test problems is needed to make any inference from a computational study. The limited number of real-world problems might be overcome by the use of synthetic optimization problems. To some extent, these problems could be made to resemble real-world problems with an appropriate selection of an input model for simulating problem instances (Reilly, 1999).

Research over the last 25 years or so has shown that one of the factors that influences the efficiency of solution methods is correlation among the objective function and constraint coefficients. Presumably, the coefficients in practical instances of SCP and other optimization problems are correlated. Several techniques can be used to induce correlation among the coefficients. Explicit correlation induction (ECI) is one of the methods for inducing correlation among the coefficients. ECI and other widely used methods are discussed in Chapter 2.

Some of the reasons why this research would be an important contribution in the field of synthetic optimization problems, and consequently for the optimization field as a whole, include:

- Synthetic optimization problems with known optima would certainly prove useful in evaluating the quality of solutions found by heuristics.
- Synthetic SCP problems with known optima and correlated coefficients would facilitate testing algorithms and heuristics on problems with realistic

characteristics.

- Virtually, an unlimited number of SCP test problems with known optima and correlated coefficients could be simulated.

The organization of this dissertation can be summarized as follows.

Chapter 2 briefly explains some relevant past research on correlation induction strategies and their pros and cons. The general overview of algorithms and heuristics for SCP and a few notable past works in simulating random optimization problems with known/unknown optimal solutions are also reviewed in this chapter. Chapter 3 is dedicated to explaining the detailed procedure of generating random SCP instances with known optimal solutions and correlated coefficients. Chapter 4 summarizes the design and the results of the computational demonstration performed during the research. Specifically, the performances of three greedy heuristics on simulated SCP instances with known optimal solutions and correlated coefficients are reported and analyzed. In Chapter 5, conclusions drawn from the research, as well as possible extensions for future work in this area, are reported and discussed.

CHAPTER TWO: LITERATURE REVIEW

This chapter discusses research on the generation of synthetic optimization problems with correlated coefficients. Research on SCP solution methods and generation of optimization problems with known optimal solution are also reviewed.

Experimenting with simulated test problems is not new, and test problems with correlated coefficients are becoming increasingly common. For example, researchers have generated test problems with correlated coefficients under implicit correlation induction for classical optimization problems such as the 0-1 Knapsack Problem (KP01), Generalized Assignment Problem (GAP), Capital Budgeting (or Multidimensional Knapsack) Problem and SCP. Reilly (2006a) stresses that induced correlation levels are rarely quantified. Instead, inadequate descriptors like “strong”, “weak”, “almost strongly correlated” and “inversely strongly correlated” are used to characterize the induced correlation levels.

Reilly (2006a) suggests that generating optimization test problems with correlated coefficients and conducting experiments with such problems would ultimately contribute to the science of algorithms that Hooker (1994) proposes. For generating correlated coefficients, two kinds of methods have been proposed in the literature. In the implicit correlation induction (ICI) methods, the desired correlation level is not specified;

however, some population correlation is implied by specifying parameters of the ICI problem generation method (Reilly, 1997). In the explicit correlation induction (ECI) methods, at least a desired population correlation or a joint distribution of coefficient values is specified. Procedures to induce correlation by these two techniques are discussed briefly and their pros and cons are considered.

Background

Let X and Y be random variables representing the values of two types of coefficients in simulated optimization problems. The correlation measure of interest between these random variables is the Pearson product-moment correlation given by:

$$\mathrm{Corr}(X,Y) = \frac{\mathrm{E}(XY) - \mathrm{E}(X)\mathrm{E}(Y)}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}} = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}}$$

If the marginal distributions of X and Y are fixed, $\mathrm{E}(X)$, $\mathrm{E}(Y)$, $\mathrm{Var}(X)$ and $\mathrm{Var}(Y)$ are also fixed. The only way to change $\mathrm{Corr}(X,Y)$ is to change $\mathrm{E}(XY)$, or the joint distribution of X and Y. ECI assumes the marginal distributions of X and Y are fixed. If, on the other hand, the marginal distributions of either or both random variables are altered as happens in ICI methods, then the resulting correlation level may also be altered.

Hill and Reilly (2000a) report on a technique for generating correlated coefficients and quantifying the correlation structure based on a multivariate composite

distribution. In this research, however, emphasis will be on bivariate distributions of coefficient values only, one for cost coefficients and another for the column sums of the constraint-coefficient matrix, **A**.

Implicit Correlation Induction (ICI) Methods

Reilly (1997) defines an ICI method as a problem simulation scheme that induces correlation among coefficient types through a user's specification of parameters for the problem generation procedure. He has quantified correlation levels which otherwise have only been qualified for various ICI generation methods used to simulate classical optimization problems. Reilly (2006a) also provides formulas to determine ICI parameters that would approximate target population correlation levels, thereby enabling some control over the target correlation in ICI problem generation methods.

Much attention has been paid to the performance of solution methods on KP01 instances with correlated coefficients. KP01 has the following form:

$$\text{Maximize } \sum_{j=1}^{n} p_j x_j$$

$$\text{Subject to } \sum_{j=1}^{n} w_j x_j \le b,$$

$$x_j \in \{0,1\},\ j = 1,2,...,n$$

where all $p_j > 0$, all $0 < w_j < b$ and $\sum_{j=1}^{n} w_j x_j > b$.

Martello and Toth (1979, 1997), Martello, Pisinger and Toth (1999, 2000) and Pisinger (1997) report computational results for simulated KP01 problems on the basis of qualified correlation levels like "uncorrelated", "weakly correlated", "value independent", "almost strongly correlated", "inversely strongly correlated" and "strongly correlated". They implement an ICI method, which can be described as follows.

Let α be a positive integer and δ and γ be nonnegative integers. Let W_j be the random variable representing constraint coefficient values and P_j be the random variable representing objective coefficient values for KP01 instances. Then, the KP01 simulator looks like:

$$w_j \leftarrow W_j \sim U\{1,2,...,\alpha\}$$
$$t_j \leftarrow T_j \sim U\{-\delta,-\delta+1,.....\delta\}$$
$$p_j \leftarrow P_j \sim w_j + t_j + \gamma$$

Reilly (1997) provides a formula for the implied population correlation level for the coefficient generation technique shown above:

$$\text{Corr}(W_j, P_j) = \sqrt{\frac{\alpha^2 - 1}{\alpha^2 + 4\delta(\delta+1) - 1}}.$$

Martello and Toth (1979) use several combinations of δ and γ to yield different (qualified) correlation levels in KP01 instances. Reilly (1997) shows that the coefficient correlation is very strong (at least 0.97) for the KP01 instances that Martello and Toth (1979) and others classify as "weakly correlated". He also points out that the correlation is perfect for the instances Martello and Toth (1979) claim to be "strongly correlated". Further, it is practically impossible to have a population correlation level of zero with this

method as it requires the parameter δ to be $+\infty$ (Cario, Clifford, Hill, J. Yang, K. Yang and Reilly, 2002). The effect of changing ICI parameters on the level of correlation has been demonstrated in Reilly (1997, 2006a). He also provides an expression for approximating δ if the desired population correlation is ρ:

$$\delta \approx \frac{-1+\sqrt{1+(\alpha^2-1)(1-\rho^2)/\rho^2}}{2}.$$

Rushmeier and Nemhauser (1993) generate SCP instances with an ICI method that can be described as follows:

$$a_{ij} \leftarrow A_{ij} \sim \text{Bernoulli}(d)$$
$$s_j = \sum_{i=1}^{m} a_{ij},$$
$$c_j \leftarrow C_j \sim U\{S_j, S_j+1, ...\omega S_j\},$$

where *d* is the expected density of the binary constraint-coefficient matrix and ω is a positive integer. S_j, which is the column sum of binary constraint coefficients, is the binomial random variable with *m* trials and success probability *d.* For this correlation induction strategy, Reilly (2006a) quantifies the population correlation level between the objective function coefficients and the column sums in the constraint matrix as:

$$\text{Corr}(S_j, C_j) = (\omega+1)\sqrt{\frac{3(1-d)}{md(\omega-1)^2 - 4d\omega^2 + (\omega+1)(4\omega-4d+2)}}.$$

Additionally, he also shows that, with all other ICI parameters kept constant, increasing any one parameter decreases the implied population correlation. He also provides an expression for approximating ω if the desired population correlation is ρ for given

values of m and d :

$$\omega \approx \frac{\rho^2\left((2m+4)d-6\right)+6(1-d)+2\rho\sqrt{\rho^2+12(1-\rho^2)d(1-d)(m-1)}}{2\rho^2\left((m-4)d+4\right)-6(1-d)}.$$

ICI techniques for GAP instances and Capital Budgeting Problem instances proposed by several other authors have been analyzed and the correlation levels induced in the resulting problems have been quantified by Reilly (1997, 2006a).

Explicit Correlation Induction (ECI) Methods

With an ECI method, either a joint distribution of coefficient values is specified or a marginal distribution of values for each type of coefficient and a correlation structure are specified (Reilly, 1999). Nelson (1987) shows how to construct bivariate probability distributions for dependent random variables with arbitrary marginal distributions and a feasible correlation level. If $g^+(a,c)$ and $g^-(a,c)$ are the maximum-correlation and minimum-correlation joint distributions for (A,C), respectively, then two classes of composite distributions for (A,C) are:

$$\lambda g^-(a,c)+(1-\lambda)g^+(a,c) \qquad (2.1)$$

where $0 \le \lambda \le 1$, and

$$(1-\alpha-\beta)f_A(a)f_C(c)+\alpha g^-(a,c)+\beta g^+(a,c) \qquad (2.2)$$

where $\alpha, \beta \geq 0$, $\alpha + \beta \leq 1$, and $f_A(a)$ and $f_C(c)$ are the marginal distributions of A and C, respectively. These composite distributions are convex combinations of the extreme-correlation distributions for (A, C), and in the case of (2.2), the joint distribution under independence. Distribution (2.1) is sometimes referred to as an extreme mixture, whereas distribution (2.2) is referred to as a conventional mixture when either α or β is equal to zero (Hill and Reilly, 2000b).

If ρ^- and ρ^+ are the theoretical minimum and maximum correlations possible for (A, C), respectively, then families of distribution with any correlation ρ such that $\rho^- \leq \rho \leq \rho^+$ are possible from the composite distribution (2.2). The composite distribution (2.2) can be rewritten as:

$$\lambda_0 f_A(a) f_C(c) + \lambda_1 g^-(a,c) + \lambda_2 g^+(a,c), \qquad (2.3)$$

where $\lambda_0 + \lambda_1 + \lambda_2 = 1$ and $\lambda_i \geq 0$ for $i = 0,\ 1,$ and 2.

When $\lambda_1 = 0$ in (2.3), the distribution represents a positively correlated bivariate random variable with correlation $\lambda_2 \rho^+$. Similarly, when $\lambda_2 = 0$, the distribution represents a negatively correlated bivariate random variable with correlation $\lambda_1 \rho^-$ (Nelson, 1987).

Peterson and Reilly (1993), Reilly (1993, 1994), Hill and Reilly (1994) and Cario et al. (2002) refer to (2.3) as a parametric mixture. This composite distribution (2.3) has been used as the basis to generate SCP coefficients in this research. For in-depth knowledge of ECI methods, interested readers are referred to the research cited above.

ECI Procedure for SCP Instances

Let A and C be the random variables with distributions representing the column sums of the constraint matrix and the objective function coefficients, respectively. Let $f_A(a)$ and $f_C(c)$ denote the marginal distributions for the random variables A and C, and let $g^-(a,c)$ and $g^+(a,c)$ be the minimum-correlation and maximum-correlation joint distributions of A and C, respectively. A procedure for SCP generating coefficients from the composite distribution shown above is:

a. Generate $u_1, u_2, u_3 \sim U(0,1)$

b. If $u_1 \le \lambda_0$ then generate $a_j \leftarrow F_1^{-1}(u_2)$ and $c_j \leftarrow F_2^{-1}(u_3)$ independently.

c. If $\lambda_0 < u_1 \le \lambda_0 + \lambda_2$, then generate $a_j \leftarrow F_1^{-1}(u_2)$ and $c_j \leftarrow F_2^{-1}(u_2)$ using common random numbers (CRN).

d. Otherwise, generate $a_j \leftarrow F_1^{-1}(u_2)$ and $c_j \leftarrow F_2^{-1}(1-u_2)$ using antithetic random numbers.

<u>Comparison of ICI and ECI Methods</u>

In general, ICI and ECI methods provide great opportunities to simulate test problems for testing optimization algorithms and heuristics. A thorough empirical evaluation should be based on a variety of test problems. Not enough real problems are

available for testing (Moore et al., 1990). Sampling under independence only would not be sufficient to determine the quality of a solution procedure. Reilly (1991) suggests that the coefficients in practical optimization problems may not be probabilistically independent. Therefore, the ICI and ECI methods certainly have opened new possibilities where a huge number of very large random problems with a wide range of correlation levels may be generated.

Following the work by Reilly (2006a) on various ICI methods suggested by many authors for different discrete optimization problems, users not only can determine the level of correlation but also can have fairly good control over the induced level of population correlation under ICI. However, ICI methods have some drawbacks and Reilly (1997) warns users to have sufficient familiarity with characteristics of practical problem instances and characteristics of problem instances generated under ICI. It is practically impossible to have zero correlation in an ICI method (Cario et al., 2002). Under ICI, population correlation levels can be altered only by altering the parameters of an ICI method which, in turn, also alters the marginal distributions of the coefficients values. Therefore, the confounded effects of problem generation parameters and correlation on algorithm performance should be considered while drawing inferences from computational experiments. Most ICI methods, as presented in the literature, generate either positively correlated or negatively correlated coefficients, but not both. Exceptions include Cario et al., (2002) and Reilly (2006b). Although there are some shortcomings in the ICI methods, they are very easy to implement and effective at inducing correlation. They are already widely used in research.

Some of the drawbacks encountered in ICI methods are remedied in ECI methods. One cannot only specify the required population correlation level, but also can systematically control it. The marginal distributions of the random variables need not be changed to alter the population correlation. An infinite number of composite distributions is possible for the same marginal distributions in an ECI method based on (2.3). There is no confounding effect of marginal distributions and correlation level, so it is easier to determine the effect of correlation alone on solution method performance. Additionally, it is possible and straightforward to simulate instances for a wide range (ρ^- to ρ^+) of correlation levels with ECI methods.

Algorithms and Heuristics for SCPs

SCP is well-known to be an NP-hard optimization problem (Karp, 1972). Many researchers have suggested algorithms and heuristics for unicost SCP instances and non-unicost SCP instances. Unicost SCPs are those SCPs where all of the cost coefficients in the objective functions are all one or any other common positive value. The SCP instances generated in this research are non-unicost instances only because correlation has no meaning in unicost instances of SCP.

There are many solution procedures, algorithms and heuristics, suggested and tested for SCP. A bibliography of heuristics and algorithms developed through the 1980s is mentioned in Ceria, Nobili, and Sassano (1997). They broadly categorize all the

heuristics as greedy heuristics, Lagrangian heuristics, or local heuristics. They also categorize exact algorithms in terms of their use of optimization approaches such as cutting-planes, branch and bound, branch and cut, and polynomially-solvable cases.

A survey of more recent algorithms and heuristics which have been computationally evaluated is given in Caprara, Toth, and Fischetti (2000). The authors classify solution procedures broadly under linear programming relaxation, heuristic procedures, and exact algorithms. Most of the heuristics and algorithms use some kind of relaxation with a slight variation in optimization techniques. For greedy heuristics alone, there are nine different criteria for selecting/ deselecting subsets or decision variables. An interesting heuristic for SCP is the one by Balas and Carrera (1996). The authors report extensive test results for the problems created by Beasley (1990), Balas and Ho (1982), and other real-life problem instances. They also report that their heuristic usually found the optimal solution; however, the Lagrangian heuristic by Beasley (1990) finds better solutions for 9 out of 35 test problems.

According to Caprara et al. (2000), popular commercially available integer linear programming solvers such as CPLEX and MINTO deploy preprocessing based on heuristics by Caprara, Toth, and Fischetti (1999) and apply LP relaxation. Integer solutions are determined by using branch-and-bound techniques with smart decision rules. These solvers are very competitive, in terms of computational time, with any of the exact algorithms presented in the literature (Caprara et al., 2000). It seems that the branch-and-bound technique has been the best contender to date, at least as far as exact solution procedures for SCP are concerned.

In this research, a procedure for simulating SCP instances with known optimal solutions and specified population correlation levels between the objective function coefficients and the column sums of the constraint coefficients has been developed. This new procedure could be incorporated in any computational evaluation of the solution methods mentioned here.

Simulating Test Problems with Known Optimal Solutions

The mathematical programming community started using computers to solve optimization problems but often felt the need for a large number of test problems so that the validity of the software codes as well as the efficacy of competing algorithms could be determined. O'Neill (1982) shows that test problems could be generated randomly by perturbing the problem data of real-world problems. He presents methods to obtain randomly generated analogs of the real-world problems by randomizing the Boolean image (0 or 1 element of the constraints matrix) or by perturbing the problem data (randomizing the elements that are greater than 1 in the constraint matrix) of real-world linear programming (LP) problems. O'Neill (1982) suggests that randomly generated analogs of the real-world problems could be used to test the efficacy of the software codes written for any LP algorithms. However, his technique of generating random problems is not able to generate problems with known optimal solutions and sometimes the generated problems are even infeasible.

One of the most studied problems in combinatorial optimization is the Traveling Salesman Problem (TSP). Arthur and Frendewey (1988) generate symmetric and asymmetric TSPs with known optimal tours using Karush-Kuhn-Tucker (KKT) conditions for equivalent 'assignment problem relaxations'. This avoids subtours in the assignment problem solutions.

Pilcher and Rardin (1992) also design a problem generation approach based on the KKT conditions for the TSP. Their approach includes selection of the optimal solution, randomly generating non-negative dual variables, and finally computing cost coefficients such that all KKT conditions are satisfied. As the objective function coefficients are not fixed prior to the generation of a problem instance, there is great flexibility in choosing the values for the objective function coefficients to force the problem instance to have all KKT conditions satisfied. Neither Arthur and Frendewey (1988) nor Pilcher and Rardin (1992) attempt to induce correlation among the coefficients.

The present research focuses on generating SCPs with known optima and specified target (population) correlation between the objective function coefficients and the corresponding column sums of the constraint matrix. The procedure developed here offers great promise for evaluating SCP algorithms and heuristics on large instances with practical characteristics.

CHAPTER THREE: METHODOLOGY

The objective of this research is to devise a procedure for generating synthetic SCP instances with known optimal solutions and specified target population correlation between the objective function coefficients and the column sums of the matrix of binary structural constraint coefficients. This chapter explains the procedure that has been developed for this purpose. It defines some terms that have been used in the procedure, explains notations, and discusses observations made during the development of the methodology. It also highlights the various features and functioning of the software program coded for this procedure.

A typical SCP can be mathematically represented as follows. The decision variables are:

$$x_j = \begin{cases} 1 \text{ if subset } j \text{ is included in the cover} \\ 0 \text{ otherwise} \end{cases}$$

for $j \in J = \{1, 2, 3, ..., n\}$. Then, the objective function and the constraints may be expressed as:

$$\text{Minimize } \sum_{j \in J} c_j x_j$$

$$\text{Subject to} \quad \sum_{j \in J} a_{ij} x_j \geq 1 \text{ for } i \in I,$$

$$x_j \in \{0,1\}, j \in J,$$

where $a_{ij} \in \{0,1\} \ \forall \ i \in I, j \in J$.

The procedure that is developed for simulating SCP instances is based on the Karush-Kuhn-Tucker conditions. It can be seen that generating SCP instances with specified population correlation among the coefficients complicates the simulation process.

Karush-Kuhn-Tucker Conditions for SCP

SCP is a pure binary integer program. The linear programming relaxation of SCP, SCP_R, is:

$$\text{Minimize} \quad \sum_{j \in J} c_j x_j$$

$$\text{Subject to} \quad \sum_{j \in J} a_{ij} x_j \geq 1 \text{ for } i \in I \quad (3.1)$$

$$-x_j \geq -1, \forall j \in J \quad (3.2)$$

$$x_j \geq 0, \forall j \in J \quad (3.3)$$

For the dual of SCP_R, there would be m dual variables denoted by $\pi_1, \pi_2, \cdots\cdots, \pi_m$ for constraints (3.1) and n dual variables denoted by $\lambda_1, \lambda_2, \ldots.., \lambda_n$ for constraints (3.2). The

dual of SCP_R is:

$$\text{Maximize} \qquad \sum_{i \in I} \pi_i - \sum_{j \in J} \lambda_j$$

$$\text{Subject to} \qquad \sum_{i \in I} a_{ij} \pi_i - \lambda_j \le c_j \quad \forall\, j \in J, \qquad (3.4)$$

$$\pi_i \ge 0 \;\; \forall \; i \in I, \qquad (3.5)$$

$$\lambda_j \ge 0 \;\; \forall \; j \in J \qquad (3.6)$$

The Karush-Kuhn-Tucker (KKT) necessary and sufficient conditions for optimality for SCP_R and its dual are:

- Primal feasibility (Eqs. 3.1, 3.2, and 3.3)
- Dual feasibility (Eqs. 3.4, 3.5, and 3.6)
- Complementary slackness (Eqs. 3.7, 3.8, and 3.9 given below)

The complementary slackness conditions (CSC) in this case may be expressed as follows:

$$\left(\sum_{j \in J} a_{ij} x_j - 1 \right) \pi_i = 0, \;\; \forall \; i \in I, \qquad (3.7)$$

$$\left(1 - x_j\right) \lambda_j = 0 \;\; \forall\, j \in J, \qquad (3.8)$$

$$x_j \left(\sum_{i \in I} a_{ij} \pi_i - \lambda_j - c_j \right) = 0 \;\; \forall\, j \in J, \qquad (3.9)$$

Condition (3.9) will sometimes be denoted in matrix form as $(\boldsymbol{\pi}\mathbf{A_j} - \lambda_j - c_j)\, x_j$ for convenience.

SCP Generation Procedure Overview

This section explains the procedure to generate SCP instances with correlated coefficients. For SCP, correlation between objective function coefficients and the corresponding column sums has been induced by Rushmeier and Nemhauser (1993) using an ICI method. Moore et al. (1990) simulate SCP instances with an ECI (conventional mixture) approach. In this research, the correlation induction strategy used is ECI and the optimal solutions are known as well. The problem generation procedure is based on the KKT conditions and the Strong Duality Theorem, which states that, if complementary solutions to the primal (that is SCP_R) and the dual problems are feasible, the solutions to both the primal and dual problems are optimal solutions.

Terminology and Notation

Some of the terminology and notations that have been used in this procedure are:

$A_j =$ $\sum_{i \in I} a_{ij}$, the sum of the j^{th} column of the binary matrix of structural constraints coefficients.

$\pi_i =$ Dual variable corresponding to the i^{th} constraint of SCP_R.

$\lambda_j =$ Dual variable corresponding to the upper bound constraint of x_j.

$J^* =$ Set of positions (indexes) of 1s in an optimal solution vector $\mathbf{x}^*$. If the optimal

solution $\mathbf{x}^*$ is (1 0 1 0 1 0), then J^* is {1, 3, 5}.

n_k^* = Number of constraint matrix columns such that $j \in J^*$ and $A_j = k$. For example, $n_2^* = 3$ means there are three columns $j \in J^*$ with $A_j = 2$.

n_k = Number of constraint matrix columns such that $A_j = k$. For example, $n_5 = 3$ means there are three column sums equal to 5.

$m_{\min}$ = Minimum number of structural constraints possible for a given optimal solution and simulated coefficients.

$m_{\max}$ = Maximum number of structural constraints possible for a given optimal solution and simulated coefficients.

m = Actual number of structural constraints that satisfies $m_{\min} \le m \le m_{\max}$.

Furthermore, $m = |\ I\ |$. Note that $n_k \ge n_k^* \ \forall\ k$.

Four-Phase SCP Generation Procedure

This procedure generates random SCP instances with a known optimum and explicitly induced correlation among the objective function coefficients and column sums of constraint coefficients. In the first phase, the coefficients are generated, an optimal solution is selected, and the number of structural constraints is chosen. The constraint columns for variable with optimal value 1 are constructed in the second phase. The values of the dual variables are assigned in the third phase. In the final phase, the

columns for variables with optimal value 0 are constructed. The phases are designed so that no trivial preprocessing will effectively reduce the size of the instance. The schematic flow diagram of this procedure is shown in Figure 1. Each of the phases of this procedure is described in the following sections.

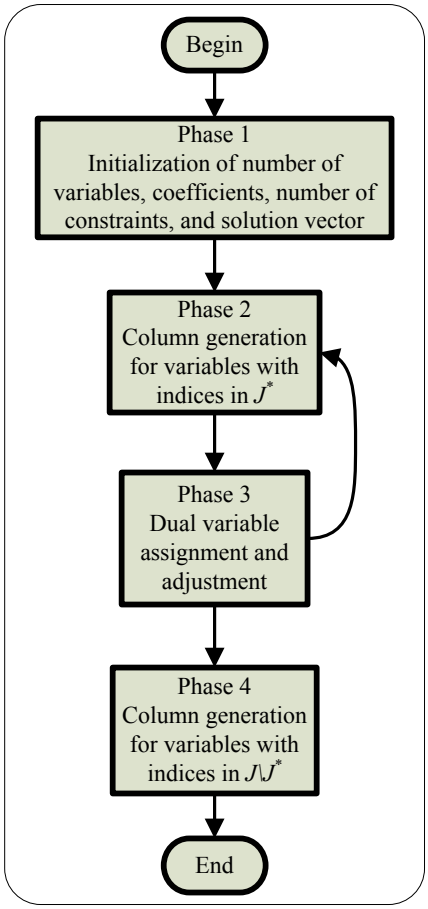


Figure 1: Schematic flow diagram of SCP generation procedure

Phase 1 -Initialization

1. Generate n random variates from a bivariate distribution with specified correlation and marginal distributions. One marginal distribution represents the values of the column sums (A_j) of the structural constraints and the other distribution represents the coefficients in the objective function (c_j).
2. Generate a binary column vector $\mathbf{x}^*$ with dimension n. Every element of this vector represents the optimal value of a decision variable. One way to generate $\mathbf{x}^*$ is to simulate n Bernoulli trials with probability of success p (i.e., $p = \Pr(x_j = 1)$ for all j). The expected proportion of decision variables with value 1 in the optimal solution is then p. (Another option would be to draw np elements from the set J without replacement. In this case, the proportion of decision variables with value 1 in the optimal solution is exactly p).
3. Select a feasible number of structural constraints m such that $m_{\min} \le m \le m_{\max}$. The minimum and maximum possible numbers of constraints can be calculated using the following formulas:

$$m_{\min} = (2k_{\max} - 1) + \sum_{k:n_k^*>0} (n_k^* - 1) - \sum_{k:n_k^*=0} [1]$$

$$m_{\max} = \sum_{j \in J^*} A_j$$

where $k_{\max} = \max_{j \in J^*} \{A_j\}$ and $k_{\min} = \min_{j \in J^*} \{A_j\}$. The justification of these formulas is explained in the Discussion section in this chapter.

4. Initialize n_k, $k = 1, 2, ..., \max_j \{A_j\}$. If $\binom{m}{k} \geq n_k$ for every value of n_k, go to Step 5. Otherwise discard $\mathbf{x}^*$ and go to Step 2.

5. Initialize every element of the constraint matrix, $a_{ij} = 0 \ \forall \ i \in I$ and $j \in J$. With this Phase 1 terminates.

It is recommended to check whether or not, for the selected number of constraints and the column sums, it is possible to generate the columns uniquely. Step 4 of the initialization phase checks for every n_k, whether there exists enough possible configurations of 1s. If there is an insufficient number of possible combinations of 1s for any n_k, the procedure regenerates the solution vector **x***.

The schematic diagram of the initialization phase is shown in Figure 2.

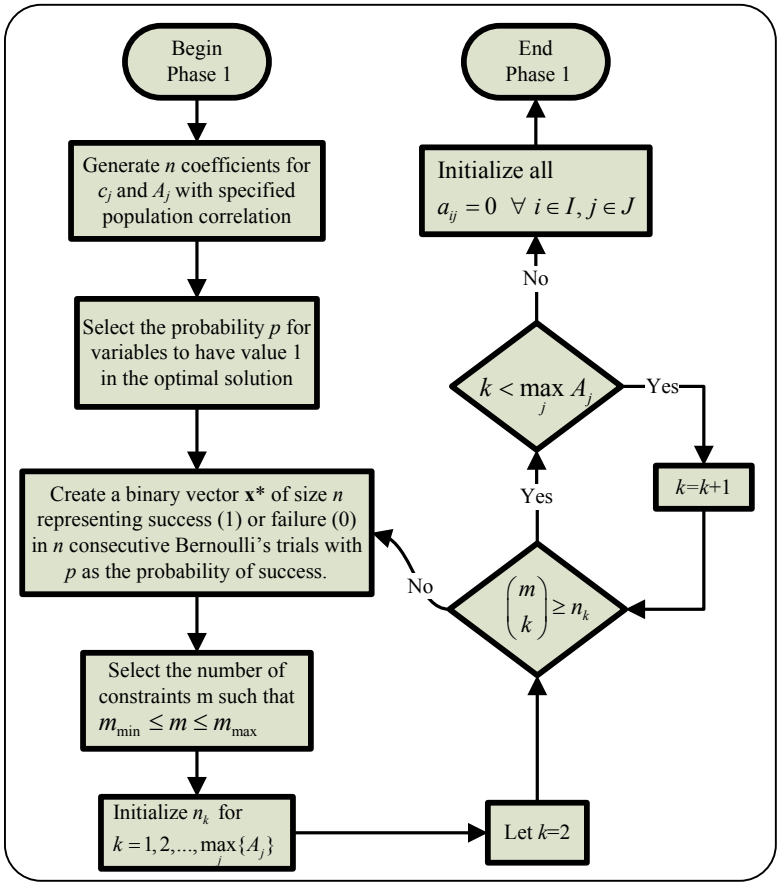


Figure 2: Schematic diagram for initialization phase

Phase 2 - Column generation for variables with optimal value 1

Here are some additional definitions and notations used to explain this procedure.

Notation:

$J \setminus J^* = J - J^*$ is the set of indices of variables that have value 0 in the optimal solution

vector.

$A^* = \sum_{j \in J^*} A_j$ = sum of A_j s for all $j \in J^*$.

$A_j^{'}$ = the remaining number of 1s still to be assigned in column j, $j \in J^*$, during the column generation process. (At the beginning of the process, $A_j^{'} = A_j$.)

$A_r = \sum_{j \in J^*} A_j^{'}$. (At the beginning of the process, $A_r = A^*$.)

f_i, $i \in I$, is the Boolean value (true/false) for row i, that indicates whether or not the corresponding row is a potential recipient of a 1 in column j, $j \in J^*$. (At the start of the column generation for $j \in J^*$, f_i = 'true' for all i.)

I_u = the set of indices of rows that are uniquely covered by exactly one $j \in J^*$, so that $| I_u | = | J^* |$. (In order to avoid further consideration of these rows for other $j \in J^*$, their f_i values are set to 'false'.)

$s_i = \sum_{j \in J} a_{ij}$ for all i. (At the beginning of this phase, $s_i = 0$ for all i. After the generation of a column, s_i is updated for all i accordingly and used to verify whether the generated column is a valid column. The generated column is valid if the remaining columns, j, $j \in J^*$, can be generated covering all the constraints with at least one constraint covered uniquely. After the generation of all columns j, $j \in J^*$, s_i will equal the left hand side value of constraint i.) Also, $s_i' = s_i$, $\forall\ i$. If it is determined that a newly generated column is valid, then the s_i' are updated with the s_i.

Define $I_0 = \{i \mid i \in I \setminus I_u \text{ and } s_i = 0\}$, i.e., the row indices of the currently violated constraints, $I_1 = \{i \mid i \in I \setminus I_u \text{ and } s_i = 1\}$, i.e., the row indices of the currently binding constraints, and $I_2 = \{i \mid i \in I \setminus I_u \text{ and } s_i > 1\}$, i.e., the row indices of the constraints that are currently non-binding and satisfied.

For any variable $j \in J^*$, only $A_j - 1$ 1s are to be assigned in the rows whose indices are in $I \setminus I_u$. Let *G* be the set which stores the possible row indices for $j \in J^*$ during column generation where a_{ij} s can be assigned 1s in the rows $i \in I \setminus I_u$. Basically, *G* can be considered a sampling bin where $A_j - 1$ numbers of row indices from $I \setminus I_u$ are stored and if they are found to be valid (i.e., $A_r \geq |I_0|$), then assignments $a_{ij} = 1$ are performed for all $i \in G$. Otherwise, *G* is discarded and another search for a valid *G* is carried out. The condition $A_r \geq |I_0|$ ensures that there are enough 1s associated with variables $j \in J^*$, that have not yet been considered, left to be assigned to the rows.

Let I_0', I_1' and I_2' be the exact replicas of I_0, I_1, and I_2 respectively, created right before column generation for some $j \in J^*$. After the generation of any column, I_0, I_1, and I_2 are updated based on the s_i if the column is valid; otherwise, I_0, I_1, and I_2 are restored to their original values that were stored prior to the column generation as I_0', I_1' and I_2'.

Let ℓ be the number of rows that are to be considered for generation of a column pertaining to any $j \in J^*$. At the beginning of column generation $\ell = m$. When any

column $j \in J^*$ is to be generated, row 1 is considered and if $f_1 = 'true'$ for that row, a_{1j} is assigned value 1 with probability as $A_j^{'} / \ell$. If a_{1j} is assigned value 1, both $A_j^{'}$ and ℓ are decreased by 1. Otherwise, only ℓ is decreased by 1. If $f_1 = 'false'$, then only ℓ is decreased by 1 and row 2 is considered for the assignment and the process continues until $A_j^{'}$ becomes zero.

Let b be the current number of constraints that are binding. Similarly, let b^c be the current number of constraints that are satisfied but not binding. Define $b_{\max}^c = \min\left\{m_{\max} - m, m - |J^*|\right\}$ as the maximum number of non-binding rows that can be generated by the procedure.

Phase 2 assigns 1s in such a way that all the rows are covered by at least one $j \in J^*$ and also every $j \in J^*$ covers at least one unique row. At the end of the generation, the number of non-binding rows is adjusted to have the maximum possible number of non-binding rows. Since the dual variable corresponding to any non-binding row is 0, this condition guarantees that 1s can be assigned in that row for any $j \in J \setminus J^*$ without violating the dual constraints. Although this adjustment is optional, the generation procedure adopts this scheme so as to limit the number of unsuccessful trials to generate valid SCP instances.

The procedure for the column generation for the variables with optimal value 1 is outlined below. Refer to Figure 5 for a schematic diagram of this procedure.

1. The following values and sets are created and initialized.

i. Initialize all $f_i =$ '*true*', $\forall\ i$.

ii. Initialize all $s_i = 0$.

iii. Assign $A_r = A^*$.

iv. Initialize $I_u = \phi$.

2. Consider the first value in J^* and let this value be j.

3. Assign $i = 1$ and $\ell = m$.

4. If $f_i = true$, go to Step 5. Otherwise, assign $i = i+1$, $\ell = \ell$ -1 and go to Step 4.

5. Generate $u \sim U(0,1)$. If $u > A_j / \ell$, assign $i = i+1$, $\ell = \ell$ -1 and go to Step 4.

   Otherwise,

   i. $a_{ij} = 1$

   ii. $f_i =$ '*false*'

   iii. $s_i = s_i + 1$

   iv. $I_u = I_u \cup \{i\}$

   v. $A_r = A_r - 1$

6. If all indices in J^* have been considered, go to Step 7. Otherwise let j be the next value in J^* and go to Step 3.

7. Determine the set $I \setminus I_u$. Create sets I_0, I_1, I_2 based on s_i and let $s_i' = s_i$, $\forall\ i$.

8. Consider the first value in J^*. Let this value be j.

9. Create the sets $I_0' = I_0$, $I_1' = I_1$ and $I_2' = I_2$.

10. Execute the function columnGenerate( ). Refer to Figure 3 for a schematic flow diagram for function columnGenerate( ).This function is a search technique which ensures random assignments of 1s for $j \in J^*$. The sub-steps of this function are:

i. Consider the first value in $I \setminus I_u$. Let this value be i.

ii. Assign $G = \phi$, L= $|I \setminus I_u|$ and $A_j^{'} = A_j - 1$. $A_j^{'}$ is decremented by 1 each time the procedure finds a valid row ($f_i = 'true'$) for $j \in J^*$.

iii. Check if $f_i = 'true'$.

a. If yes, generate $u \sim U(0,1)$. If $u \leq A_j^{'} / \ell$, then $G = G \cup \{i\}$, $\ell = \ell$ -1, $A_j^{'} = A_j^{'}$ -1, $s_i = s_i + 1$ and go to Step 10-iv. Otherwise, go to Step 10-iii-b.

b. Else, $\ell = \ell$ -1, consider the next value in $I \setminus I_u$. Let this value be i and go to Step 10-iii.

iv. Check if $A_j^{'} > 0$.

a. If yes, consider the next value in $I \setminus I_u$. Let this value i and go to Step 10-iii.

b. Else, this procedure terminates.

11. Update I_0, I_1 and I_2 based on $s_i, i \in I$.

12. If $A_r - A_j + 1 \geq |I_0|$, $a_{ij} = 1 \ \forall \ i \in G$, $s_i' = s_i$, $A_r = A_r - A_j + 1$ and go to Step 13. Otherwise, restore I_0, I_1, I_2 and s_i based on I_0', I_1', I_2' and $s_i^{'}$, respectively. Let $G = \phi$ and go to Step 10.

13. Check if $A_r = |I_0|$

i. If yes,

a. Assign $f_i = '\,false\,' \;\forall\; i \in I_1 \cup I_2$.

b. Check if there is any value in J^* yet to be considered.

- If yes, go to Step 13-ii.
- Else, go to Step 14.

ii. Otherwise, proceed as follows,

a. Consider the next value in J^*. Let this value be j.

b. Update the sets I_0', I_1' and I_2' based on I_0, I_1, and I_2, respectively, and go to Step 10.

14. Check if $b^c = b^c_{\max}$

a. If yes, column generation for $j \in J^*$ terminates.

b. Else, go to Step 15.

15. This step executes the function nbRowAdjustment( ). Refer to Figure 4 for the schematic flow diagram for this function. This function searches for the column covering the largest number of non-binding rows and assigns a 1 in one of the corresponding non-binding rows. A 1 is removed from the row with the largest row sum and it is reassigned in the same column it was removed from but in one of the non-binding rows. Details of the procedure for this function are outlined below.

i. Determine a column index pertaining to a variable which covers the maximum number of non-binding rows. If there is a tie, the one with the smaller c_j is

chosen. Let this column index be $j^{\max}$.

ii. Determine the row with the largest row sum. Let this row be r.

iii. Create a list of column indices such that $a_{rj} = 1$ and $j \neq j^{\max}$ and randomly select any value from the list. Let this value be d.

iv. Along column $j^{\max}$ randomly choose row k such that $a_{kj^{\max}} = 1$ and $k \in I_1$.

v. Assign $a_{kd} = 1,\ a_{rd} = 0,\ s_k = s_k + 1$ and $s_r = s_r - 1$.

vi. Update I_0, I_1 and I_2 based on the new s_i.

vii. If $b^c = b^c_{\max}$ the function terminates. Otherwise, go to Step 15.

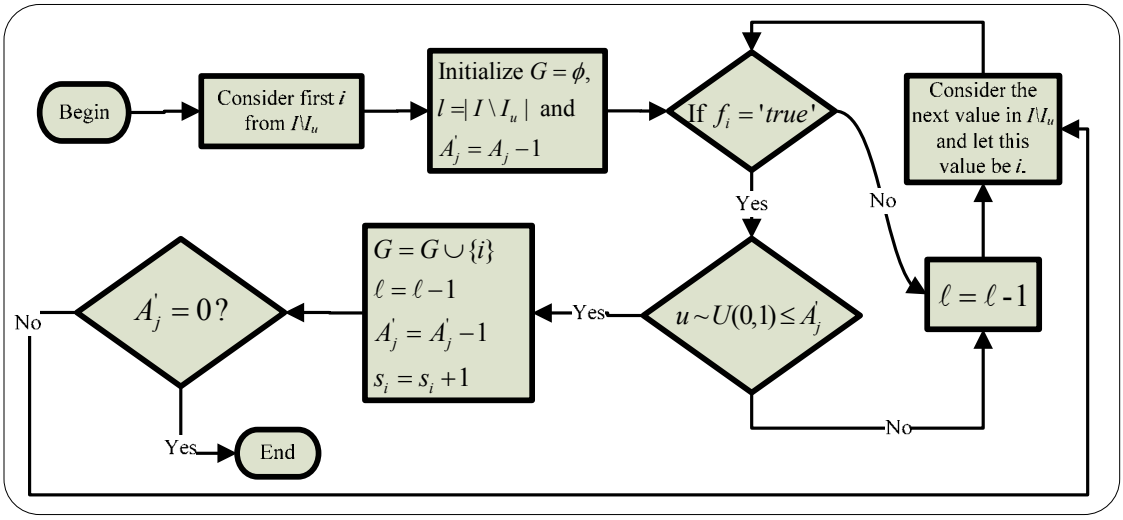


Figure 3: Schematic diagram for the function columnGenerate( )

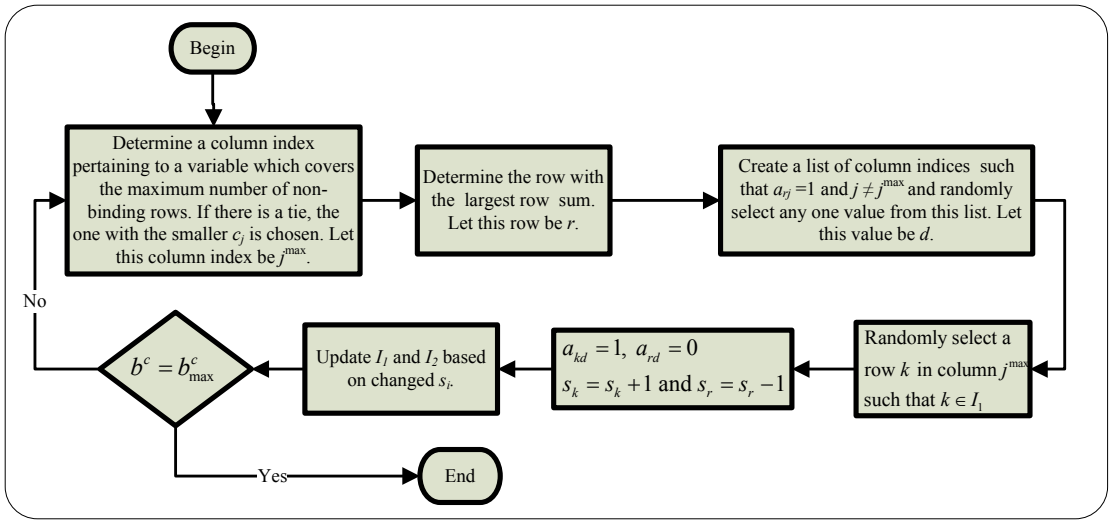


Figure 4: Schematic diagram for function nbRowAdjustment( )

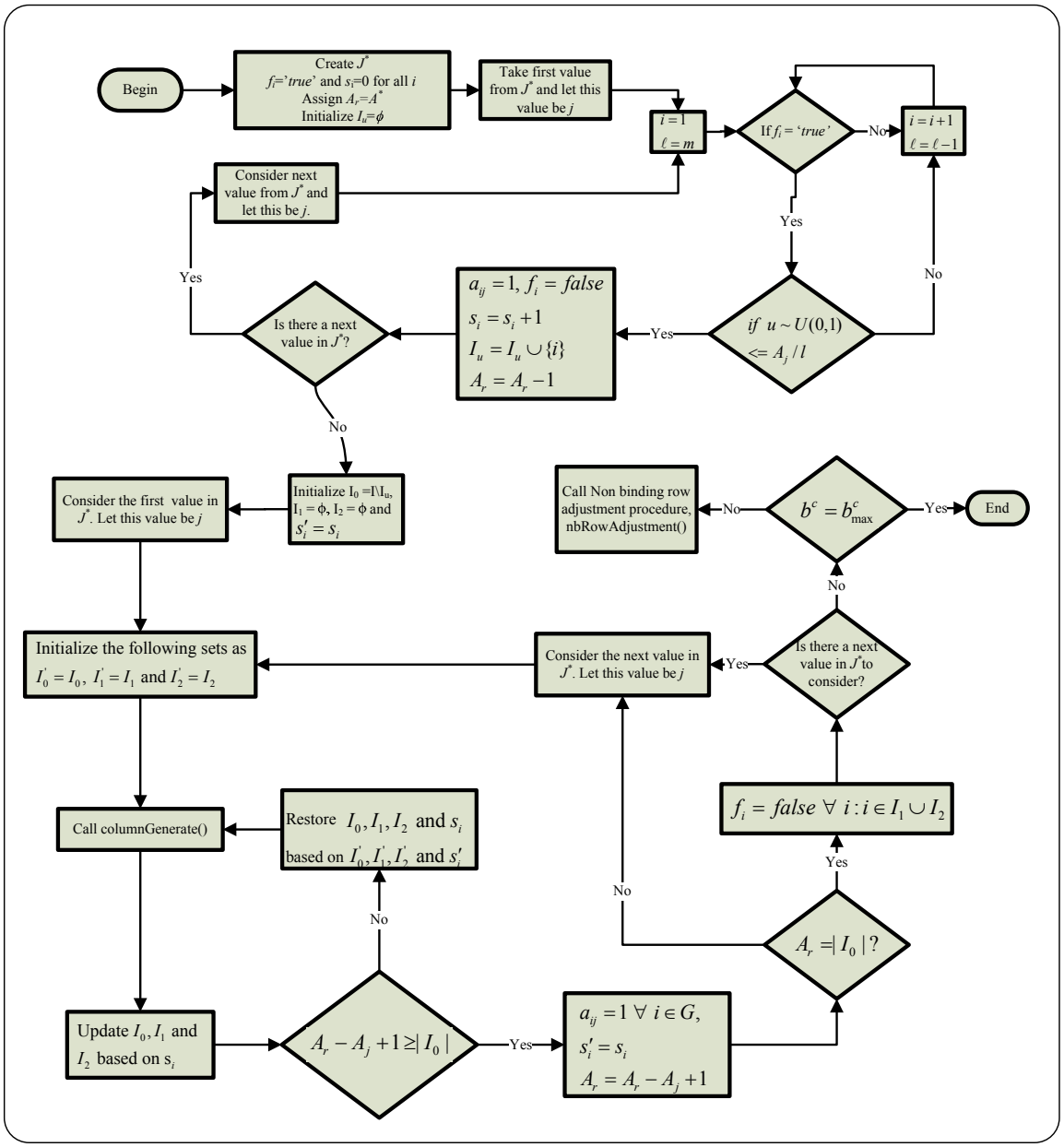


Figure 5: Schematic diagram for generation of columns $j\in J^*$

Phase 3 - Dual variables assignment and adjustment

In this third phase, dual variable values are assigned. Then these dual variables are checked for the possibility of generating a valid SCP instance. If an adjustment is required, the values of these variables are adjusted. The adjustment of dual variables arises in two conditions: either it is impossible to assign 1s for at least one $j \in J \setminus J^*$ and maintain $\boldsymbol{\pi}\mathbf{A_j} - \lambda_j \leq c_j$, or there are not enough configurations of 1s so as to have unique columns. A column is considered unique if it is somehow different from the rest of the columns with the same A_j. The dual variable assignment and adjustment procedures are explained below. Refer to Figure 6 for a flowchart of the dual variable assignment procedure.

Dual variable assignment procedure

1. Assign $\pi_i = 0 \ \forall \ i \ni s_i > 1$. Since only columns pertaining to $j \in J^*$ have been generated $s_i = \sum_{j \in J^*} a_{ij}$ for all i. Therefore, s_i is the value of the left hand side of constraint i.
2. Consider the first value in J^* and let this value be j.
3. Define a set M_j, which stores the row indices such that a_{ij}=1 and s_i=1 . (The set M_j stores the rows indices of the binding constraints covered by x_j).

4. Assign $\pi_i = \frac{c_j}{|M_j|}$ $\forall\ i \in M_j$. For example if $j \in J^*$, c_j = 65 and M_j =3, then

   $\pi_i = \frac{65}{3}$ for all $i \in M_j$.

5. Check if there is another value in J^* to consider.

   i. If yes, consider the next value in J^* and let this value be j. Go to Step 3.

   ii. Otherwise go to Step 6.

6. Create a separate list of ordered dual variables $\pi_{<i>}$ such that $\pi_{<1>} \le \pi_{<2>} \le ... \le \pi_{<m>}$.

7. Create another list based on the list created in Step 6. Every value in this second list is given by $\sum_{r=1}^{i} \pi_{<r>}$. (For example, in the list of $\sum_{r=1}^{i} \pi_{<r>}$, entry '5' will have the number that is the sum of first five dual variables in the list of the $\pi_{<i>}$.)

8. The assignment procedure terminates.

The assignment strategy explained above assumes $\lambda_j = 0$ for columns $j \in J^*$. KKT condition (3.8), requires that λ_j =0 for all $j \in J \setminus J^*$ to satisfy $\left(1 - x_j\right)\lambda_j = 0 \ \ \forall\ j \in J$. However, λ_j for column, $j \in J^*$ can have any non-negative value if KKT condition (3.9), $x_j\left(\sum_{i \in I} a_{ij}\pi_i - \lambda_j - c_j\right) = 0 \ \ \forall\ j \in J,$ is satisfied. In doing so, the dual variables π_i s corresponding to the binding rows covered by $j \in J^*$ column will be greater as, $\sum_{i \in I} a_{ij}\pi_i = \lambda_j + c_j$ to satisfy (3.9). In the case where $\lambda_j = 0$, $\sum_{i \in I} a_{ij}\pi_i = c_j$, and π_i s

corresponding to the binding rows covered by $j \in J^*$ column will be relatively smaller. During the generation of column $j \in J \setminus J^*$, (3.9) is automatically satisfied whereas the dual constraint $\boldsymbol{\pi}\mathbf{A_j} - \lambda_j \leq c_j$ should also be satisfied with the valid configuration of 1s. Any generated column $j \in J \setminus J^*$ is valid if and only if the column is not the replica of another column and the dual constraint is satisfied. In the case of larger values of π_i s , the number of possible combinations of 1s along the columns $j \in J \setminus J^*$ is reduced compared to the case with relatively smaller values of π_i s (i.e., case where $\lambda_j = 0$). This scenario may lead to the case where valid columns $j \in J \setminus J^*$ may not be generated and regeneration of columns $j \in J \setminus J^*$ should be performed, beginning with the new values for the dual variables.

The scheme of assigning $\lambda_j = 0$ is for all $j \in J$ is not mandatory. As explained earlier, the objective here is to minimize the number of regeneration of dual variables by maintaining higher probability of generating valid columns $j \in J \setminus J^*$.

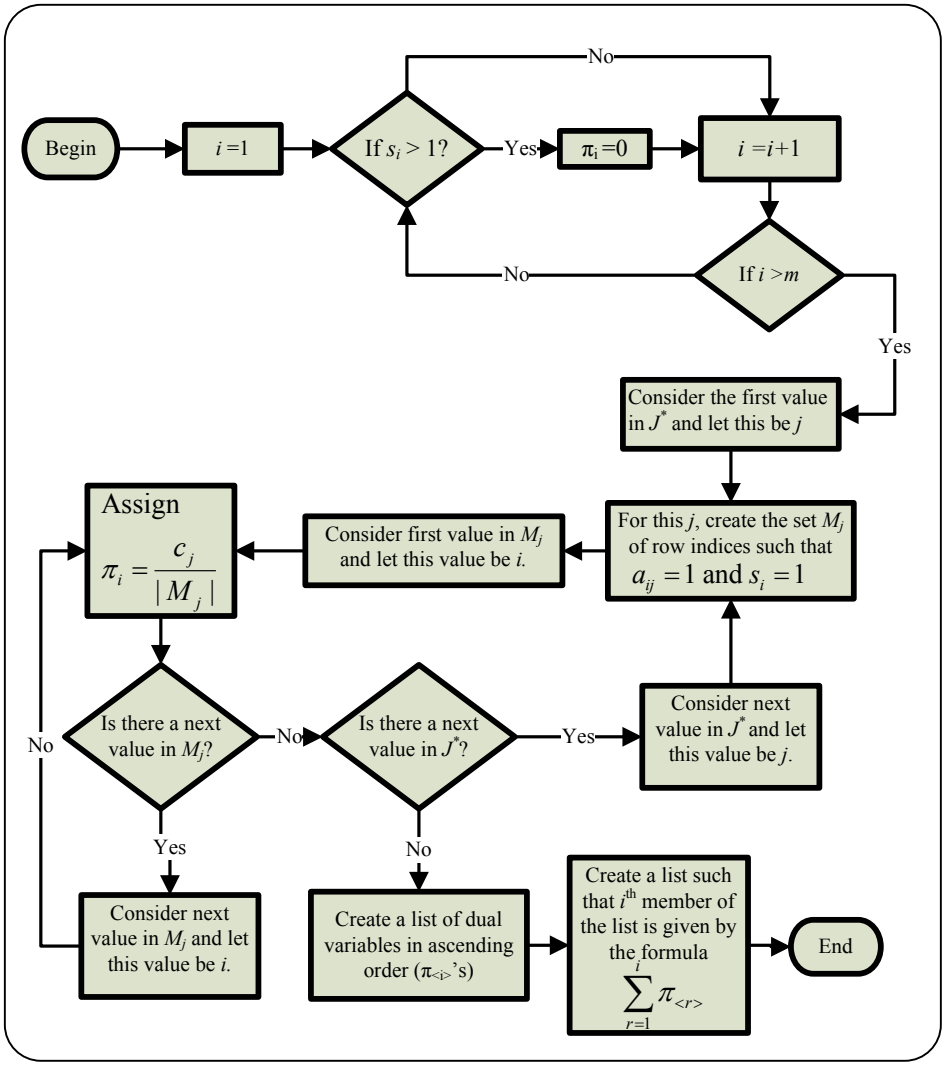


Figure 6: Dual variable assignment procedure

Consider a problem with 50 variables and 25 constraints. Suppose that for the selected solution vector, 11 and 29 are the theoretical minimum and maximum numbers of constraints possible. After the assignment of dual variables as mentioned above, $\pi_{<i>}$, $\sum_{r=1}^{i} \pi_{<r>}$ and $< i >$ are as follows:

Table 1 Example of $\pi_{<i>}$ and their ranks

i	$\pi_{<i>}$	$\sum_{r=1}^{i} \pi_{<r>}$	$< i >$
5	0	0	1
6	0	0	2
9	0	0	3
11	0	0	4
12	4	**4**	5
13	4	8	6
4	19	27	7
7	19	46	8
8	19	65	9
10	19	84	10
17	20	**104**	11
20	22	126	12
24	22	**148**	13
25	23	171	14
14	33	204	15
15	33	237	16
18	33	270	17
19	33	303	18
1	37	340	19
2	37	377	20
3	38	415	21
16	43	458	22
21	43	501	23
22	43	544	24
23	48	592	25

The actual row in the constraint matrix that each dual variable corresponds to is the first column denoted by i in Table 1 and the last column $< i >$ is the ranking number for $\pi_{<i>}$.

Some useful information can be drawn from Table 1:

- There are four rows that have more than one 1 in columns $j \in J^*$. Though all these extra 1s can go in one dense row and still yield a valid problem instance, spreading these 1s over as many rows as possible makes many rows inactive, thereby forcing

the corresponding rows to have dual variables = 0.

- The entries in the $\sum_{r=1}^{i} \pi_{<r>}$ column give the least possible c_j for $j \in J \setminus J^*$ with $A_j = <i>$ to satisfy the dual constraints. For example, $\sum_{r=1}^{5} \pi_{<r>} = 4$ means, if any $j \in J \setminus J^*$ with $A_j = 5$ has $c_j < 4$ in the SCP instance being generated, then it is impossible to satisfy the dual constraints pertaining to that x_j. Furthermore, an SCP instance cannot be generated if $j \in J \setminus J^*$ and $A_j = 2$, 3 or 4 and $c_j < 0$ (which is not encountered as all $c_j > 0$).
- For any $j \in J \setminus J^*$, the number of different configurations of 1s possible can be deduced. For example, let us consider a partial list of decision variables and their coefficients shown in Table 2.

Consider x_{14}, which has $c_4 = 124$ and $A_4 = 4$. In Table 1, 104 is the largest value of $\sum_{r=1}^{i} \pi_{<r>}$ less than or equal to 124. This means that there are 11 rows that can have 1s allowing $\binom{11}{4}$ ways to configure those four 1s. Similarly, for x_2, there are $\binom{13}{5}$ configurations of 1s possible. Checking all columns $j \in J \setminus J^*$ for the possibility of acquiring a unique column has to be ensured prior to generation of columns $j \in J \setminus J^*$.

Table 2 A partial list of cost coefficients and column sums

j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
c_j	67	157	167	130	50	14	53	52	48	140	169	8	18	124	15	114	116	36	112	185
A_j	4	5	5	4	3	3	3	3	3	5	5	5	2	4	2	2	4	5	4	5

Generally, the adjustment of dual variables is very likely needed when a problem instance is created for the maximum possible number of constraints because all the dual variables assume non-negative values since none of the constraints are non-binding. This kind of configuration of dual variables may make assignment of 1s for some columns $j \in J \setminus J^*$ impossible without violating the dual constraints.

The columns pertaining to $j \in J \setminus J^*$ have a typical characteristic when a problem instance is created with the maximum number of constraints. Every row is binding, ensuring non-negative values for the corresponding dual variables. As per the assignment procedure for dual variables explained earlier, in this case, every dual variable corresponding to some row assumes value c_j / A_j where $j \in J^*$ covers that row, thereby making $x_j \left(\sum_{i \in I} a_{ij} \pi_i - \lambda_j - c_j \right) = 0$. Upon completion of the assignment of dual variables, if there is no difficulty in assigning 1s for all columns pertaining to $j \in J \setminus J^*$ and maintaining their uniqueness, the adjustment of dual variables is not necessary. Otherwise, a procedure for checking and adjusting the dual variables is implemented as outlined below.

Dual variable checking and adjustment procedure

Some new notation used in this procedure includes:

$\bar{k}_{\max} = \max_{j \in J} \{A_j\}$

q = maximum number of adjustments of dual variables needed for any column $j \in J \setminus J^*$ in order to satisfy the dual constraint $\boldsymbol{\pi}\mathbf{A_j} - \lambda_j \leq c_j$.

w = the number of binding rows covered by a column $j \in J^*$.

$\bar{k}$ = A_j corresponding to the column $J \setminus J^*$ that demands q adjustments.

1. Consider the n_k s initialized earlier in Phase 1 Step 4. As defined earlier, n_k stores the count of $A_j = k$ for all $j \in J$, where $k = 1, 2, \cdots, \max_j \{A_j\}$. It is also possible to have other $n_k = 0$ for k such that $2 \leq k \leq \bar{k}_{\max}$.
2. Create a list of $c_k^{\min}$ which stores Min c_j for $A_j = k$. (Obviously, $c_1^{\min} = 0$ as the generation procedure does not consider columns with A_j =1.)
3. Initialize $q = 0$ and $\bar{k} = 0$.
4. Determine the value of q as follows:
   i. Assign $k = 2$.
   ii. Find the largest $< i >$ such that $\sum_{r=1}^{i} \pi_{<r>} \leq c_k^{\min}$. If the first value in the $\sum_{r=1}^{i} \pi_{<r>}$ column is greater than $c_k^{\min}$, then $< i >$ =0.

iii. Calculate the possible number of configurations of 1s as $\binom{<i>}{k}$.

a. If $\binom{<i>}{k} \geq n_k$, then check if $k = \bar{k}_{\max}$.

- If no, assign $k = k+1$ and go to Step 4-ii.
- Otherwise go to Step 5.

b. Otherwise, assign $q = \max\{q, (k - <i> + 1)\}$, $\bar{k} = k$ and check if $k = \bar{k}_{\max}$.

- If yes, assign $k = k+1$ and go to Step 4-ii.
- Otherwise go to Step 5.

5. If $q = 0$ go to Step 9, otherwise, change the dual variable as follows.

i. Create a list Z of column indices such that $j \in J^*$ and x_j covers more than two binding constraints and is sorted in descending order of number of binding constraints covered. If there is a tie, then the j with smaller c_j comes first in the list. Check if $Z = \phi$,

a. If yes, go to Step 5-v.

b. Otherwise, go to Step 5-ii.

ii. Let the first value in Z be z_1. Determine the number of binding rows w corresponding to the column z_1.

iii. If $q \leq w$-2 (that is if the number of adjustments required is less than the number of binding rows covered by x_{z_1} minus 2), randomly select q binding rows and do

the following.

a. Assign the dual variables corresponding to the rows selected above a small non-negative value say ∂. (Refer to the Discussion section of this chapter on the calculation of ∂.)

b. Reassign the value of the dual variables corresponding to the rows not selected as $(c_{z_1} - q\partial)/(w-q)$. Go to Step 9.

iv. If $q > w$-2 (number of adjustment required is greater than the possible number of adjustment of dual variables corresponding to the rows with 1s in column of (x_{z_1} - 2), randomly select w-2 binding rows and do the following.

a. Assign the dual variables corresponding to the rows selected above a small non-negative value say ∂.

b. Reassign the value of the dual variables corresponding to the rows not selected as $(c_j - \partial(w-2))/2$. Decrease q by $(w-2)$.

c. $Z = Z - \{z_1\}$.

d. Check if $Z = \phi$

- If yes, go to Step 5-v.
- Otherwise, go to Step 5-ii.

v. Create a list Z of column indices such that $j \in J^*$ covers exactly two binding constraints in ascending order of c_j. If $Z = \phi$ go to Step 5-viii, otherwise, go to Step 5-vi.

vi. Let the first value in Z be z_1.

vii. Select one binding row randomly out of the two binding rows covered by column z_1 and do the following:

a. Assign the corresponding dual variable for the selected row a small value ∂.

b. Assign the dual variable corresponding to the binding row that was not selected as $c_j - \partial$.

c. Decrease q by 1.

d. $Z = Z - \{ z_1 \}$.

e. If $q > 0$ and $Z \neq \phi$ go to Step 5-vi, otherwise go to Step 5-viii.

viii. Check the following conditions.

a. If $q = 0$, go to Step 9.

b. If $q = 1$ and $Z = \phi$, go to Step 6.

c. If $q > 1$ and $Z = \phi$, go to Step 10.

6. Calculate $\sum_{r=1}^{i} \pi_{<r>}$ and $< i >$ for the adjusted dual variables.

7. Create a list of c_j s for variables with $A_j = \bar{k}$ for all $j \in J$ and sort it in descending order of c_j. Create a temporary variable h and initialize $h = n_{\bar{k}}$ (i.e., $k = \bar{k}$).

8. Consider the first value in the list above and do the following:

i. Determine the corresponding $< i >$ for the greatest value in the $\sum_{r=1}^{i} \pi_{<r>}$ column

that is less than or equal to c_j .

ii. Check if $\left(\begin{matrix} < i > \\ \overline{k} \end{matrix} \right) \geq h$,

a. If yes do the following

- Assign $h = h$ -1.
- Check if there is any value of c_j in the list created in Step 7.
    - If yes, consider next value of c_j and go to Step 8-i.
    - Otherwise, go to Step 9.

b. Otherwise go to Step 10.

9. The process of the checking and adjustment of the dual variables terminates with success.
10. Print error message "Dual Variables Assignment unachievable". Restart the generation process Phase 2 with different J^* columns.

The schematic diagram for checking and adjustment of all dual variables is shown in Figure 7 and the schematic diagram of checking the possibility of the unique configurations for columns with $A_j = \overline{k}$ is shown in Figure 8.

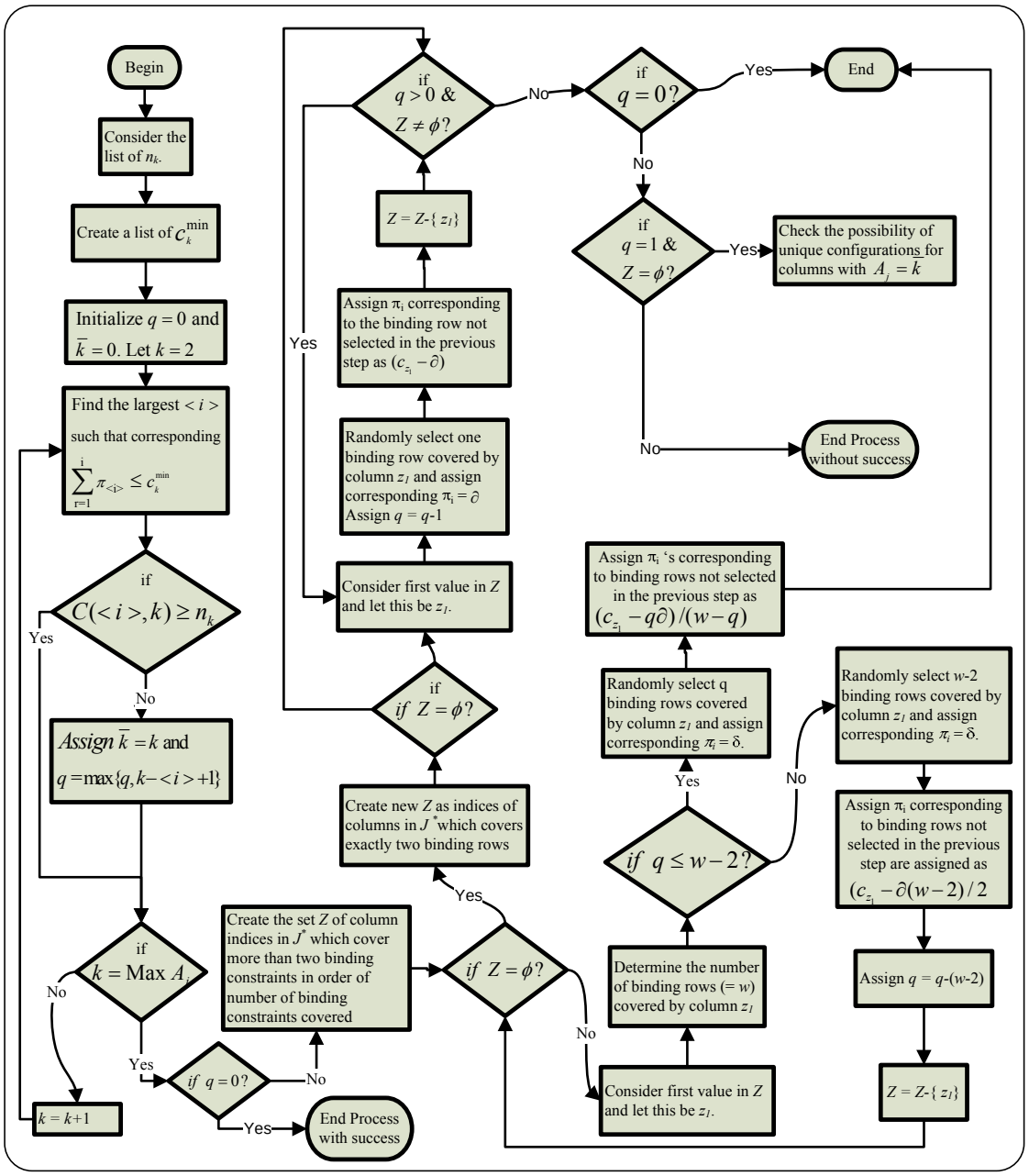


Figure 7: Schematic diagram of dual variables checking and adjustment procedure

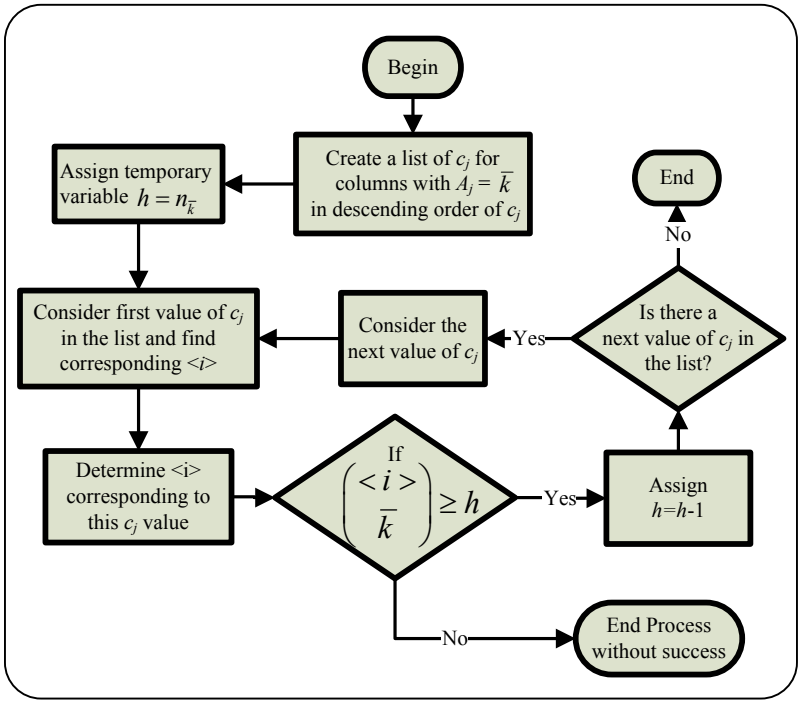


Figure 8: Checking possibility of the unique configurations for the columns with $A_j = \bar{k}$

To illustrate the assignment of values to the dual variables, an example with 17 constraints and 3 variables with optimal value 1 is presented in Table 3. For this problem, Table 4 shows that there is one variable with A_j = 2 and, with initial dual variable assignments, there are 105 unique configurations of 1s for this variable. Likewise there are 4 variables with A_j = 3 and, with these dual variable assignments, there are 330 unique configurations of 1s. The same argument is true for variables with A_j = 4 and 5 as well. However, for variables with A_j = 6, there are six variables and no possible configuration of 1s. This condition requires adjustment of the dual variables. The maximum number of adjustments of dual variables needed (q) in this case is $n_6 - <i> + 1$

= 6 – 0 +1 = 7. Therefore, seven dual variables have to be adjusted in such a way that $\pi A_j - \lambda_j \leq c_j$ is satisfied for the variable with A_j =6 and c_j = 1 (lowest). In this case that variable would be x_{20}. Table 5 shows the problem after the adjustment of 7 dual variables as per the procedure explained earlier.

Table 3 An example of the initial dual variable assignments

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	s_i	π	$\pi_{<i>}$	$\sum_{r=1}^{i} \pi_{<k>}$	$<i>$
c_j	52	193	84	64	24	140	108	130	21	12	37	32	171	114	172	62	80	96	85	1					
A_j	4	2	3	5	5	3	6	4	6	**6**	5	5	3	3	**6**	**5**	6	4	4	6					
x_j	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0					
1										0					1	0					1	28	2	2	1
2										0					0	1					1	12	2	4	2
3										0					0	1					1	12	2	6	3
4										0					0	1					1	12	2	8	4
5										1					0	0					1	2	2	10	5
6										1					0	0					1	2	2	12	6
7										1					0	0					1	2	12	24	7
8										1					0	0					1	2	12	36	8
9										0					0	1					1	12	12	48	9
10										0					0	1					1	14	12	60	10
11										1					0	0					1	2	14	74	11
12										0					1	0					1	28	28	102	12
13										1					0	0					1	2	28	130	13
14										0					1	0					1	28	28	158	14
15										0					1	0					1	28	28	186	15
16										0					1	0					1	28	28	214	16
17										0					1	0					1	32	32	246	17

Table 4 An example showing calculation of q

A_j	n_k	$c_k^{\min}$	$\sum_{r=1}^{i} \pi_{<r>}$	$<i>$	$\binom{<i>}{A_j}$	$\binom{<i>}{A_j} \geq n_k$	q
2	1	193	186	15	105	TRUE	0
3	4	84	74	11	330	TRUE	0
4	4	52	48	9	126	TRUE	0
5	5	32	32	7	21	TRUE	0
6	6	1	0	0	0	FALSE	**7**

Table 5 An example of the dual variable adjustments

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	s_i	π	$\pi_{<i>}$	$\sum_{r=1}^{i} \pi_{<k>}$	$<i>$
c_j	52	193	84	64	24	140	108	130	21	12	37	32	171	114	172	62	80	96	85	1					
A_j	4	2	3	5	5	3	6	4	6	**6**	5	5	3	3	**6**	**5**	6	4	4	6					
x_j	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0					
1										0					1	0					1	57.2	0.1	0.1	1
2										0					0	1					1	12	0.1	0.2	2
3										0					0	1					1	12	0.1	0.3	3
4										0					0	1					1	12	0.1	0.4	4
5										1					0	0					1	0.1	0.1	0.5	5
6										1					0	0					1	5.8	0.1	0.6	6
7										1					0	0					1	0.1	0.1	0.7	7
8										1					0	0					1	5.8	5.8	6.5	8
9										0					0	1					1	12	5.8	12.3	9
10										0					0	1					1	14	12	24.3	10
11										1					0	0					1	0.1	12	36.3	11
12										0					1	0					1	0.1	12	48.3	12
13										1					0	0					1	0.1	12	60.3	13
14										0					1	0					1	57.2	14	74.3	14
15										0					1	0					1	0.1	57.23	131.5	15
16										0					1	0					1	57.2	57.23	188.8	16
17										0					1	0					1	0.1	57.23	246.0	17

Table 6 Checking possibility of the unique configurations for the columns with A_j=6

	A_j	c_j	$\sum_{r=1}^{i} \pi_{<r>}$	$<i>$	$\binom{<i>}{A_j}$	Minimum number of configurations required	Flag
1	6	172	131.5	15	5005	6	Ok
2	6	108	74.3	14	3003	5	Ok
3	6	62	60.6	13	1716	4	Ok
4	6	21	12.3	9	84	3	Ok
5	6	12	6.5	8	28	2	Ok
6	6	1	0.6	7	7	1	Ok

Table 6 shows how the checking procedure works. Each column is checked in descending order of c_j. For example, in the first line, it shows that there are 5005 different configurations of 1s possible and the minimum number of configurations of 1s required is 6 (because that is the frequency of the variables with A_j = 6). Similarly the second line shows that there are 3003 different combinations of columns possible and the minimum number of required combinations is 5. When the calculated number of combinations is greater than or equal to the required number of combinations then one can proceed with checking of the remaining variables. If, by any chance, this condition is not met, then the process is terminated and regeneration of the problem beginning with Phase 2 should be performed.

Phase 4 - Column generation for variables with optimal value 0

In order to satisfy the dual constraints for columns $j \in J \setminus J^*$, the configuration of

1s should be such that the condition $\boldsymbol{\pi}\mathbf{A_j} - \lambda_j \leq c_j$ is satisfied for all $j \in J \setminus J^*$. For such columns $x_j = 0$ and consequently $\lambda_j = 0$, so in effect, the dual constraint becomes $\boldsymbol{\pi}\mathbf{A_j} \leq c_j$. Independent Bernoulli trials are performed with the success probability equal to the ratio of the column sum to the number of structural constraints of the primal problem which have dual variables less than or equal to the corresponding c_j. In order to achieve a column with exactly A_j 1s, the success probability is updated depending upon whether a success was achieved in the previous trial or not.

Consider a situation where a column to be generated for $A_j = 4$ and the number of constraints = 25. In this case, the first trial will have probability of A_j/25 or 4/25 that the a_{1j} will be assigned value 1. If this trial fails, then the probability for the next trial would be 4/24. But if this trail succeeds, then the probability for the next trial would be 3/24. This procedure continues and at the end of the 25th trial or sooner, this column will have exactly four 1s. During this process, in order to minimize the rejection of columns generated, trials for a row with $\pi_i > c_j$ should be skipped, a_{ij} should be set to 0 and the success probability for the subsequent trial should account for this assignment.

During the generation of a column, if there is a situation where $\boldsymbol{\pi}\mathbf{A_j} - \lambda_j > c_j$ then that column is discarded and a new attempt to generate a column is made. In the event that some $\boldsymbol{\pi}\mathbf{A_j} - \lambda_j = c_j$, then the corresponding x_j are potential candidates for optimal variables in alternate optimal solutions.

The schematic flow diagram of this phase is shown in Figure 9. This procedure is activated upon verifying that the dual variables have been assigned and that it is possible to generate columns for all $j \in J \setminus J^*$.

Some new concepts and notations used to explain this procedure include:

$\boldsymbol{\pi} = (\pi_1, \pi_2, ..., \pi_m)$ is a vector of dual variables, where π_i is the dual variable corresponding to the constraints i.

A stack is a special kind of arrangement of values where any new value can be put only on the top of the stack. If removal of any value from the stack is required then only the values from the top can be removed one at a time. This special characteristic is crucial in assigning 1s for column $j \in J \setminus J^*$.

The row stack is the stack of row indices where 1s for column $j \in J \setminus J^*$ can be assigned. Row stacks for different columns should differ in at least one component. An invalid stack is the stack of row indices for which 1s can not be assigned for a particular column $j \in J \setminus J^*$.

Let t_i be the Boolean value (true/false) indicating whether or not row i can be considered a candidate to accept 1s for column $j \in J \setminus J^*$. If t_i = 'false' then a_{ij} cannot be assigned value 1. Every time a new $j \in J \setminus J^*$ is considered for column generation, both the stacks are initialized as empty stacks and the t_i s are initialized as 'true'.

This column generation procedure searches for the combination of 1s and 0s for any column $j \in J \setminus J^*$ such that the column is unique and the dual constraint pertaining to

that variable is also satisfied.

Let $A_j^{'}$ be the remaining number of 1s still to be assigned for any one $j \in J \setminus J^*$ during the column generation process. (At the beginning of the process $A_j^{'} = A_j$). Define $c_j^{'}$ as c_j - $\pi \mathbf{A_j}$ for the column being generated. (At the beginning of the column generation $c_j^{'} = c_j$).

The overall process of column generation for column $j \in J \setminus J^*$ is shown in Figure 9. The procedure for column generation for variables with optimal value 0 is as follows:

1. Determine $\overline{k}_{\max}$.
2. Consider $A_j = 2$.
3. Create a set $J^{'}$ that stores the column indices of those columns which have column sum equal to A_j in the ascending order of their c_j s for all $j \in J \setminus J^*$, i.e., $j_1^{'}$ is the column with the smallest c_j among columns with column equal to A_j
4. Check if $| J^{'} | = 0$
   i. If yes, go to Step 6-ii.
   ii. Otherwise, go to Step 5.
5. Consider the first value in the set $J^{'}$ and let this value be $j_i^{'}$.
6. If $x_{j_i^{'}} \neq 1$, go to Step 7. Otherwise, check if there is a next value in $J^{'}$ to consider.
   i. If yes, let the next value in $J^{'}$ be $j_i^{'}$ and go to Step 6.

ii. Otherwise, check if the column sum of the column being considered is equal to $\bar{k}_{\max}$.

a. If yes, the process terminates as the last variable with column sum as Max A_j has already been generated.

b. Otherwise, $A_j = A_j + 1$ and go to Step 3.

7. Assign $A_j^{'} = A_{j_i}^{'}$ and $c_j^{'} = c_{j_i^{'}}$. For all rows assign t_i = *'true'*.

8. Execute function inValidate( ). (The schematic flow diagram for this function is shown in Figure 10.) This function performs the following tasks.

i. Initialize $i = 1$.

ii. If t_i = *'false'*, $i = i + 1$ and go to Step 8-iii. Otherwise, do the following:

a. If $\pi_i \le c_j^{'}$, $i = i + 1$ and go to Step 8-iii. Otherwise, do the following:

- Insert i in the invalid stack.
- Assign t_i = *'false'*.
- Assign $i = i + 1$ and go to Step 8-iii.

iii. If $i > m$, insert -1 at the top of the invalid stack and the function terminates. Otherwise, go to Step 8-ii.

9. Check if there is any π_i, such that t_i = '*true*'.

i. If yes, randomly select any one π_i with corresponding t_i = '*true*' and do the following:

a. Insert i corresponding to this π_i in the row stack

b. Assign t_i = '*false*' so that this row will not be considered again for this variable.

c. Decrease the value of $A_j^{'}$ by 1 and $c_j^{'}$ by π_i. This means, there are $A_j^{'} - 1$ 1s remaining to assign and not violate the dual constraint. The total sum of the dual variables corresponding to other rows which would be assigned $A_j^{'} - 1$ 1s later in this column, should not exceed $c_j^{'}$.

d. Go to Step 10.

ii. Otherwise, execute function reArrange( ). The schematic flow diagram for this function is shown in Figure 11. This function performs the following tasks:

a. Remove the topmost value in the invalid stack, which is -1.

b. Remove again the topmost value in the invalid stack and check if this value is -1. (There can be more than one -1 at the top of the stack)

- If not, assign t_i = '*true*' corresponding to the value removed in Step 9-ii-b and go to Step ii-b.
- Otherwise, remove the topmost value of the row stack (let this value be k) and do the following:
  - Insert k into the invalid stack.
  - $A_j^{'} = A_j^{'} + 1$
  - $c_j^{'} = c_j^{'} + \pi_k$
  - Insert -1 at the top of the invalid stack go to Step 10.

10. Check if $A_j^{'} = 0$

i. If not, go to Step 8, i.e., function inValidate( ) is executed again.

ii. Otherwise, create one temporary binary column vector of size m with all 0's except in the rows whose indices are in the row stack and go to Step 11.

11. Check for all columns whether this temporary column vector is unique compared to all the columns with $A_j = A_{j_i}$ for $j \in J$.

i. If yes, then 1s are assigned in the column $j_i^{'}$ at the rows of the primal constraint matrix whose indices are in the row stack, otherwise 0s are assigned. Discard both the invalid and row stacks. If there is a next value $j_i^{'}$ in $J^{'}$ to consider, go to Step 6. Otherwise, go to Step 6-ii.

ii. Otherwise, discard both the row and invalid stacks and go to Step 7 for the regeneration of column $j_i^{'}$.

After coefficients are generated for all the variables, a quick check of the row sums is performed. This is necessary since a row with row sum 1 must be covered by the variable which has 1 in that row. This means, by inspection one can tell that this variable should be in the optimal solution with value 1.

Under the condition where there are rows with the row sum equal to 1, a reconfiguration of 1s should be done so that every row sum is at least equal to 2. This can be done in the following manner. Let a deficient row be the row for which the row sum is less than 2. Likewise, let a donor row be the row which has the row sum greater than 2. There may be more than one possible donor row; however, a simple rule can be

established to select one row as a donor from the pool of potential donor rows.

1. Initialize the set $R = \phi$.
2. If $s_i < 2,\ R = R \cup \{i\}\ \forall\ i \in I$.
3. If $R = \phi$, go to Step 11-b. Otherwise consider the first value in R and let this value be r_1 and go to Step 4.
4. Create a list Z of the column indices in descending order of c_j / A_j for all $j \in J \setminus J^*$ except for the column which has $a_{r_1 j} = 1$. (This is because the column with $a_{r_1 j} = 1$ has already 1 in the deficient row.) Let the values stored in this list Z be $z_1, z_2, ..., z_k$. This is the list of indices of the variables which are potential donor of 1s for the deficient rows. The column z_1 having the highest c_j / A_j will have better probability of accepting 1s in the deficient row; however, the selection is restricted by the dual condition $\boldsymbol{\pi}\mathbf{A_j} - \lambda_j \leq c_j$. All i which have a_{iz_1}=1 and s_i>2 are the potential donor rows.
5. Consider the first value in Z and let this value be z_1.
6. Initialize a set of indices of donor rows $D = \phi$.
   i. Assign i=1.
   ii. If a_{iz_1}=1 and s_i>2, then $D = D \cup \{i\}$. Otherwise $i = i$+1.
   iii. If $i > m$ go to Step 7. Otherwise, go to Step 6-ii.

7. Consider the first value in *D* and let this be d_i.

8. If $\pi_{d_i} \geq \pi_{r_1}$, then make $a_{r_1 z_1} = 1$, $s_{r_1} = s_{r_1} + 1$, $s_{d_i} = s_{d_i}$ -1, and $a_{d_i z_1} = 0$. Go to step 9.

9. If $c_{z_k} - \boldsymbol{\pi}\mathbf{A}_{z_1} \geq \pi_{r_1} - \pi_{d_i}$ then make $a_{r_1 z_1} = 1, a_{d_i z_1} = 0, s_{r_1} = s_{r_1} + 1$, and $s_{d_i} = s_{d_i}$ -1 go to Step 10. Otherwise, $D = D - \{d_i\}$, check if $D \neq \phi$.

   i. If true, consider the first value in *D*. Let this value be d_i and go to Step 8.

   ii. Otherwise, $Z = Z - \{z_1\}$. If $Z \neq \phi$ consider the first value in *Z*, z_1, and go to Step 6. Otherwise go to Step 12.

10. Check if the column z_1 is unique.

   i. If yes, go to Step 11.

   ii. Otherwise, assign $a_{r_1 z_1} = 0$, $s_{r_1} = s_{r_1}$ -1, $a_{d_i z_1} = 1$, and $s_{d_i} = s_{d_i} + 1$. Check if there is an another value in *D* to consider.

      a. If yes, consider the next value in *D*. Let this value be d_i and go to Step 8.

      b. Otherwise, check if there is another value of *Z* to consider.

         - If yes, consider the next value in *Z*. Let this value be z_1 and go to Step 6.
         - Otherwise go to Step 12.

11. $R = R - \{r_1\}$ and check if $R \neq \phi$,

      a. If true, consider the first value in *R*. Let this value be r_1 and go to Step 4.

      b. Otherwise, the process terminates with success.

12. The process terminates with an error message "row-sums adjustment

unattainable".

The schematic diagram of the procedure to adjust row sums of the constraints matrix is shown in Figure 12.

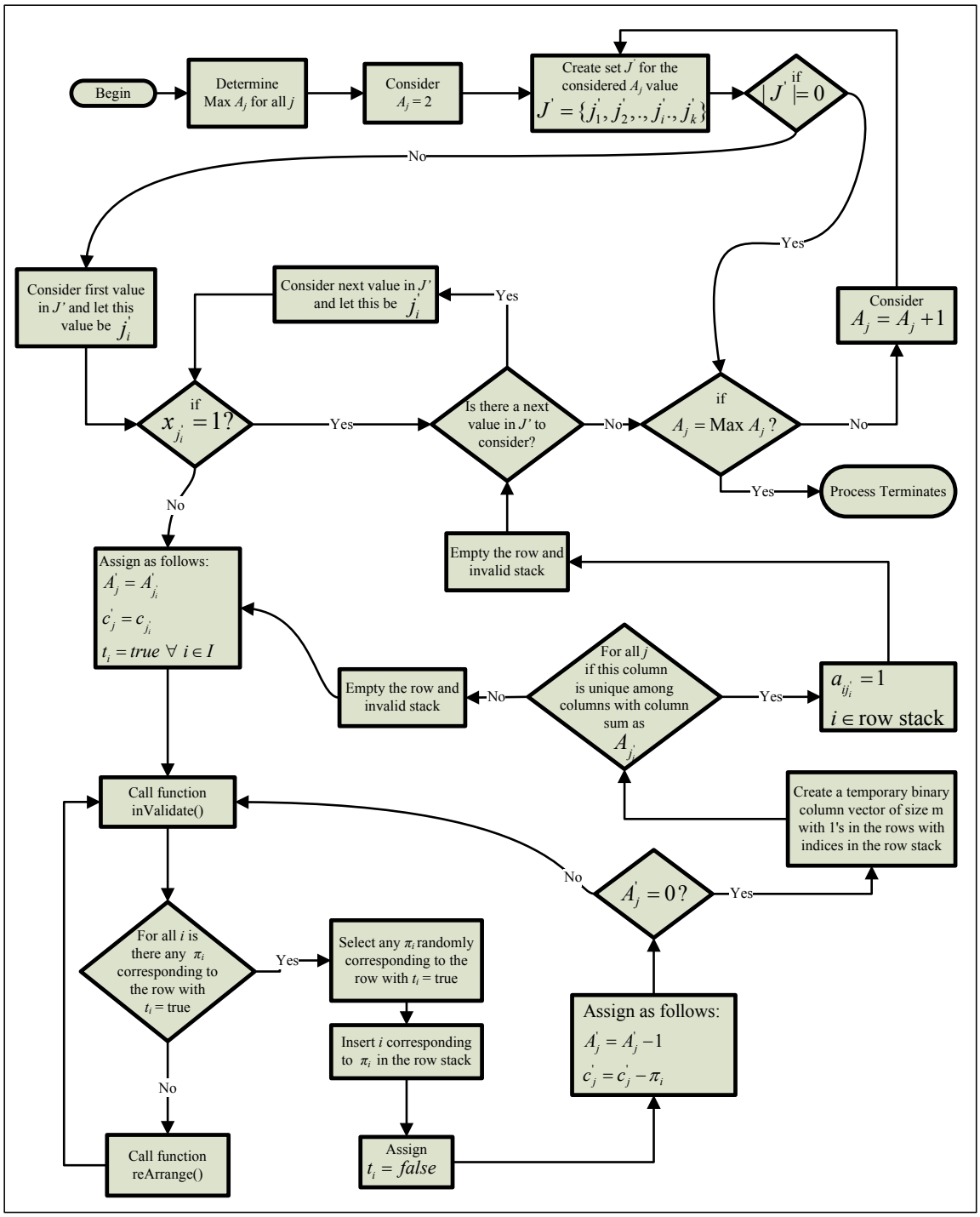
Sometimes, however the procedure explained above can not adjust the 1s along the $J \setminus J^*$ columns. In fact this procedure checks if any 1 along $J \setminus J^*$ columns can be moved to a deficient row along the same column. Generally this method is adequate to ensure row sums at least equal to 2 after adjustment. If this procedure does not work, a second row sum adjustment procedure is recommended. The second procedure differs from the earlier procedure mainly in the scope of change. The second procedure changes the entire column instead of a single element in a column during the adjustment. The detail of this procedure is explained below.

1. Initialize the set $R = \phi$.
2. If $s_i < 2,\ R = R \cup \{i\}\ \forall\ i \in I$.
3. If $R = \phi$, go to Step 8-b. Otherwise sort *R* in decreasing order of the corresponding dual variables and let the first value be r_1 and go to Step 4.
4. Create a temporary list of column indices for $J \setminus J^*$ columns such that for any column j in $J \setminus J^*$, $< i >$ is greater or equal to $A_j - 1$ where $< i >$ is the maximum value for which $\sum_{k=1}^{i} \pi_{<k>} \le (c_j - \pi_{r_1})$. As explained earlier in the dual variable assignment $\pi_{<k>}$ s are the ordered dual variables so that $\pi_{<1>} \le \pi_{<2>} \le \ldots \le \pi_{<m>}$.

5. Create a list Z of column indices of $J \setminus J^*$ columns from the temporary list created in the previous step 4 such that if these columns are assigned all zeros one column at a time does not alter the number of deficient rows.
6. If $Z = \phi$, the adjustment process terminates without success. Otherwise, consider the first value in Z. Let this value be z_1 and go the Step 7.
7. Create a new column z_1 such that one 1 is assigned in a row r_1 and remaining $(A_{z_1} - 1)$ 1s along the column are assigned in such a way that the corresponding dual constraint $\boldsymbol{\pi}\mathbf{A}_j - \lambda_j \leq c_j$ is satisfied.
8. $R = R - \{r_1\}$ and check if $R \neq \phi$,
    a. If true, consider the first value in *R*. Let this value be r_1 and go to Step 4.
    b. Otherwise, the process terminates with success.

The schematic diagram of the second procedure to adjust row sums is shown in Figure 13.

With the completion of this phase, the procedure for creating SCP instances with known optimal solution and correlated coefficients terminates. An example of this SCP instance generation procedure is illustrated in Appendix A. If the second row sums adjustment procedure also fails to adjust the row sums it is suggested to regenerate the problem. Note that the problem generated is still a valid SCP instance, however, the variables covering the rows with only one 1 should have optimal value 1. To avoid easy preprocessing it is desired to have at least two 1s in every row of constraint matrix.



80

Figure 9: Schematic diagram for generation of columns $j \in J \setminus J^*$

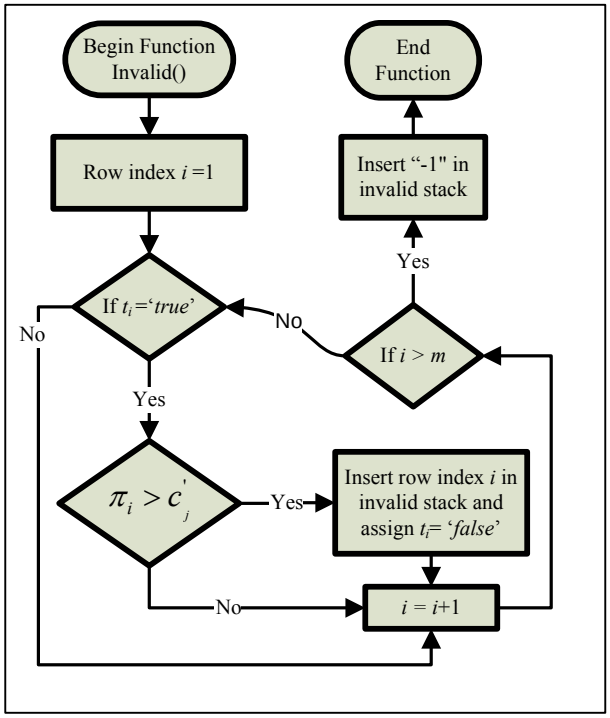


Figure 10: Schematic diagram for function inValidate( )

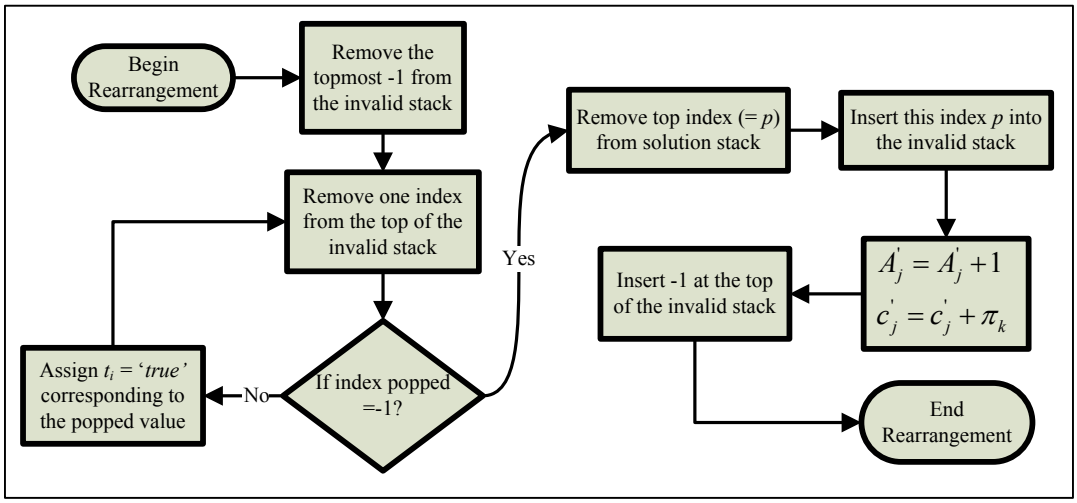


Figure 11: Schematic diagram for function reArrange( )

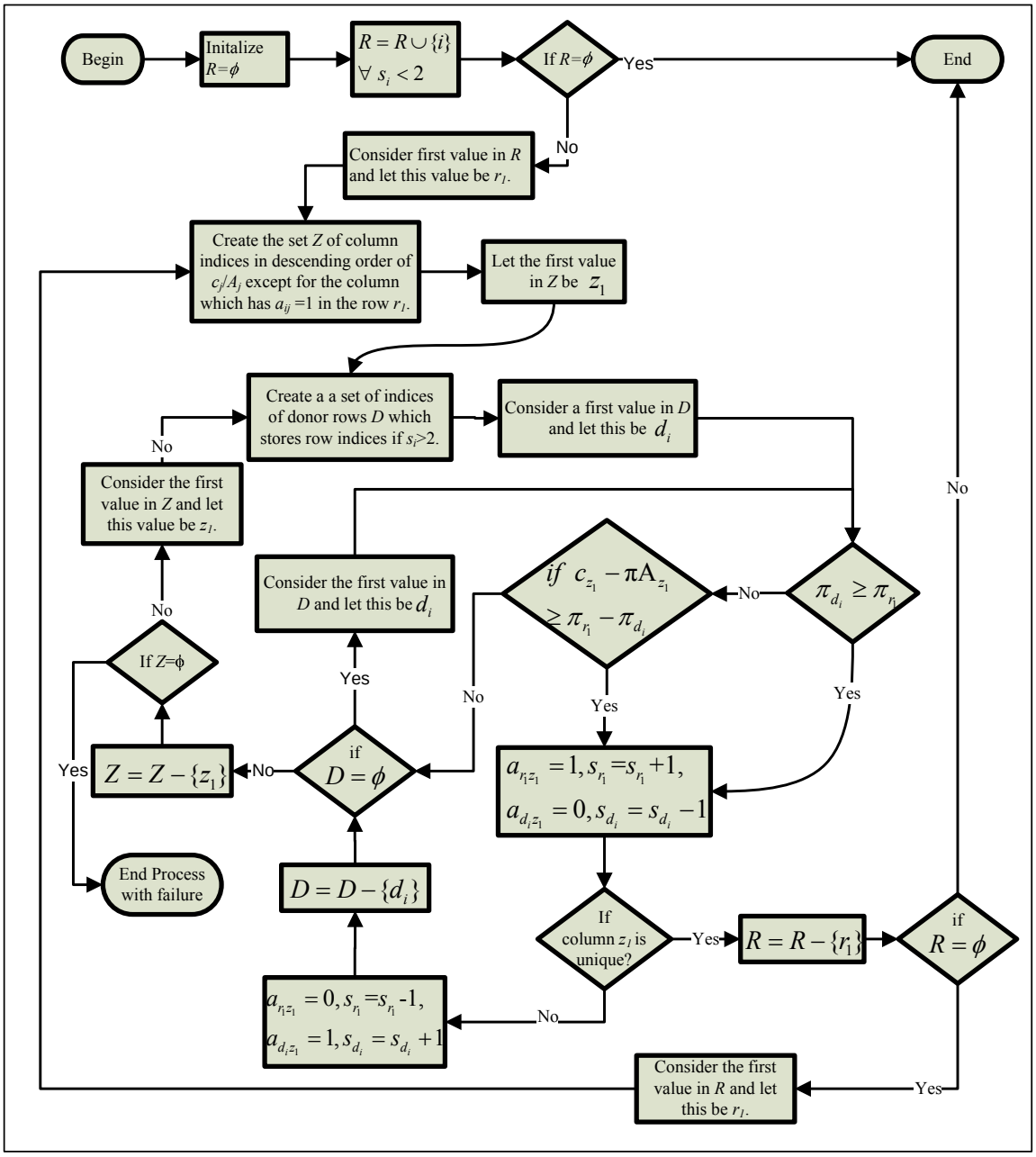


Figure 12: Schematic diagram of row sum adjustment

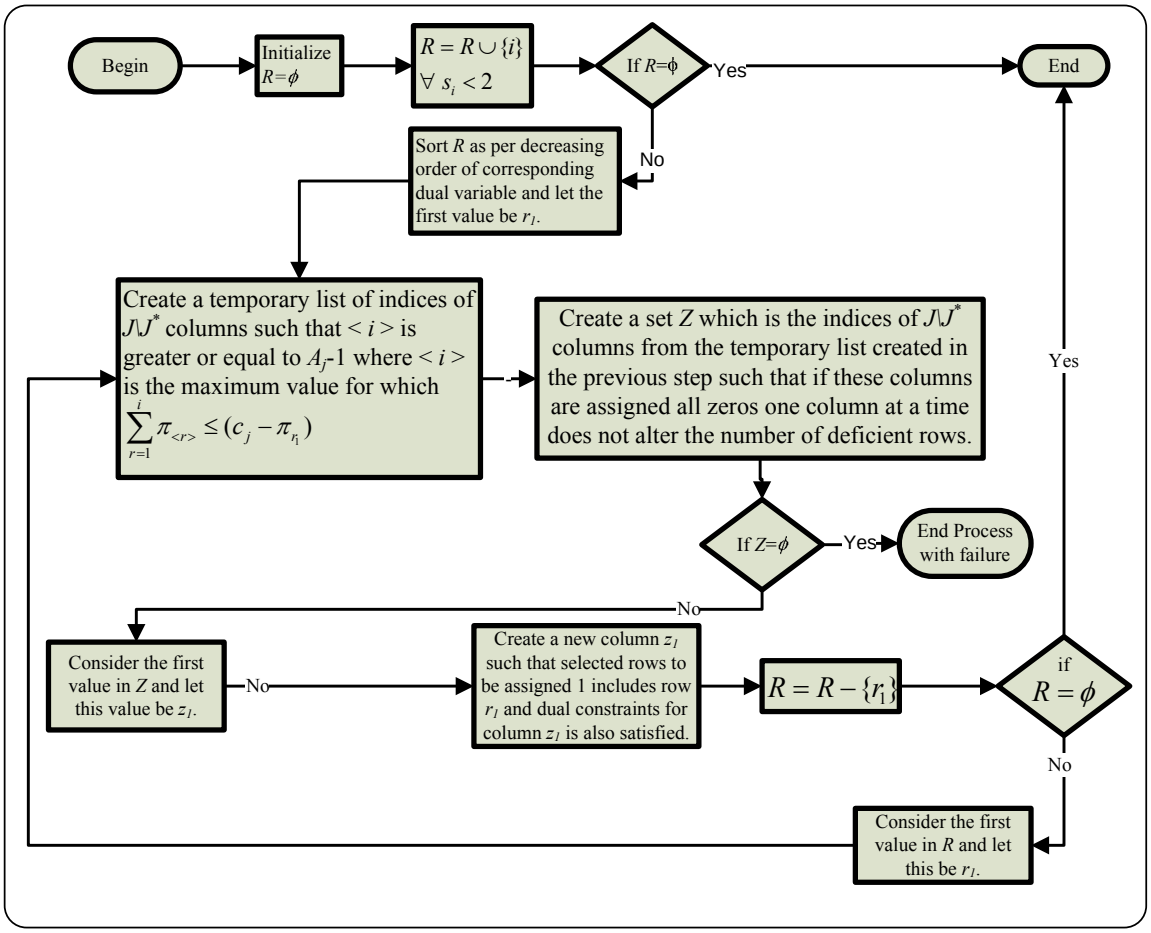


Figure 13: Schematic diagram of row sums adjustment procedure (second)

Discussion

A number of observations are made during the development phase of this procedure for generating SCP instances with known optimum solutions and correlated

coefficients. For instance, there cannot be any arbitrary number of structural constraints in an SCP instance for a given optimal solution and constraint-matrix column sums. The justification for the calculation of $m_{\min}$ and $m_{\max}$ is explained with examples. Even after the values c_j, A_j and x_j have been finalized, prior to the generation of the binary matrix **A**, a quick check of the infeasibility condition discussed below should be made. During the process of generating the binary matrix **A**, a superfluous variable condition should be checked for each column generated.

Superfluous Variable Conditions

The superfluous variable condition (SVC) occurs when one or more variables designated to have optimal value 1 have no effect on the primal feasibility of the structural constraints, i.e., one or more variables with indices in J^* can assume value 0 in the optimal solution and yet primal feasibility is unchanged. This may occur when columns $j \in J^*$ are generated randomly such that they cover all the primal constraints and any one of the columns $j \in J^*$ is effectively a combination of some other columns $j \in J^*$. The details of SVC and how to eliminate this condition during the SCP instance generation are explained in Appendix B.

Infeasibility Conditions

One type of infeasibility condition occurs when the constraint matrix for SCP cannot be generated for the given column sums, number of variables with optimal value 1, and number of constraints. Upper and lower bounds on the number of constraints are important parameters in designing a procedure for generating SCP instances with known optimal solutions. The following relation must hold true regarding the number of constraints:

$$m \leq \sum_{j \in J^*} A_j$$

In other words, $\sum_{j \in J^*} A_j$ is the upper limit for the number of constraints. If the chosen number of constraints exceeds this maximum value then primal infeasibility occurs.

A second type of infeasibility condition may occur when there are many candidate decision variables and the configuration of 1s in the constraints matrix cannot attain uniqueness. For example, a problem with m =6 cannot have A_j =2 for more than 15 variables. If it so happens, there will be columns with the same configuration of 1s in the constraint matrix. Similarly, if m =10 then, there can be at most $\binom{10}{2} = 45$ columns with A_j =2. This means, the number of variables is restricted by the values of A_j s and the value of m .

The calculation of the maximum number of columns for each discrete value of column sum can be determined by the combination formula. If there are m constraints,

then, for every column with sum A_j there are $\binom{m}{A_j}$ unique columns possible. For columns with A_j = 2, even m = 8 would allow 28 unique configurations of 1s. In the m =8 case with A_j = 3, there could be 56 ways to configure unique columns. It can be concluded from this discussion that more constraints allow more decision variables to be incorporated in the problem. Table 7 shows the maximum number of variables with different values of A_j and m .

Table 7 Maximum possible number of variables for given m and A_j

A_j	m =10	m =15	m =20	m =22	m =25
1	10	15	20	22	25
2	45	105	190	231	300
3	120	455	1140	1540	2300
4	210	1365	4845	7315	12650
5	252	3003	15504	26334	53130
6	210	5005	38760	74613	177100
7	120	6435	77520	170544	480700
8	45	6435	125970	319770	1081575
9	10	5005	167960	497420	2042975
10	1	3003	184756	646646	3268760
11		1365	167960	705432	4457400
12		455	125970	646646	5200300
13		105	77520	497420	5200300
14		15	38760	319770	4457400
15		1	15504	170544	3268760
16			4845	74613	2042975
17			1140	26334	1081575
18			190	7315	480700
19			20	1540	177100
20			1	231	53130
21				22	12650
22				1	2300
23					300
24					25
25					1

A question may arise as to whether m or n should be set first. It is suggested to set n first and check whether unique columns with the given set of A_j values may be generated for the chosen number of constraints as is done in the procedure outlined earlier in this chapter.

Calculation of the Number of Constraints

The maximum and minimum numbers of constraints are functions of the number of decision variables with optimal value 1 and their column sums. An important assumption in the derivation of the range of the number of constraints is that the decision variables with optimal value 1 should each cover at least one unique row of the constraint matrix. Setting $x_j = 0$ for any $j \in J^*$ would result in the violation of the primal feasibility of SCP. Additionally, the configuration of 1s in columns $j \in J^*$ should not encounter the SVC.

Let, $k_{\max} = \max_{j \in J^*} \{A_j\}$ as before and $k_{\min} = \min_{j \in J^*} \{A_j\}$.

The maximum number of constraints can be calculated as:

$$m_{\max} = \sum_{j \in J^*} A_j$$

The formula above suggests that each optimal variable can possibly cover as many unique rows as the corresponding column sum. For example, consider the example given

below. Assuming a problem has 5 optimal variables with the column sums as 1, 2, 3, 4 and 5. In this case, $k_{\max} = 5$ and $k_{\min} = 1$ which implies $m_{\max} = 15$. In the example matrix given below, only columns in J^* have been shown. Here, it is impossible to add even a single row that would be covered by one of the variables with optimal value 1. This means 15 (1+2+3+4+5) is the upper limit on the number of structural constraints for this example. If we forcefully add any row in the matrix below and are still able to maintain the uniqueness of columns in J^* and also are able to avoid encountering SVC condition, then we are actually formulating a problem instance with different column sums. In essence, the number of constraints in the matrix below is 15 and any endeavor to increase that number would change their column sums. This is important as this procedure of generating SCP instances uses predetermined column sums as input.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In the matrix shown above, any row or column can be interchanged to have many unique configurations.

Next various configurations of J^* are considered for determining the minimum numbers of constraints.

Consecutive and Unique Values Case

Suppose that $n_1^* = n_2^* = n_3^* = ... = n_{k_{\max}}^* = 1$.

Consider a problem with $k_{\max}$=2. Then **A** (for only $j \in J^*$) must be

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Basically, all the matrices shown above are obtained by interchanging rows or columns. (The number of possible configurations increases drastically when there are more variables or rows). Here, the minimum number of constraints is given by 1 + 2 = 3.

Now, consider a problem with $k_{\max}$ = 3. Then, **A** (for only $j \in J^*$) may be

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

There are other instances of **A** possible by swapping any two rows or columns in the

matrices shown above. However, it is evident that, in every case, n_3^* has one row covered by n_2^* such that only two rows are added with the addition of new variable with corresponding n_3^*=1. Here the minimum number of constraints is 5 (1+2+2) where the 1 represents the row corresponding to an optimal variable with A_j=1, the first 2 represents the rows corresponding to an optimal variable with A_j=2 and the final 2 corresponds to the additional rows that must be added to include a column with A_j=3.

Any variable with optimal value 1 and column sum = A_j can have a maximum of $A_j - 1$ rows with 1s such that the same rows have 1s in the columns for another optimal variable or combination of other optimal variables. With this notion, a table has been built below which shows how 1s could be configured in the J^* columns in an instance with the minimum number of constraints possible.

In Table 8, it is shown how 1s could be distributed for any $k_{\max}$ to have the minimum number of constraints when the values n_k^*s are consecutive and unique. For $k_{\max} = 2$, the first shaded cell represents a row contributed by a variable with $A_j = 1$ and the second shaded cell represents two rows contributed by another variable with $A_j = 2$. Therefore, for $k_{\max} = 2$, all rows are uniquely covered by two variables. For $k_{\max} = 3$, with the addition of a variable with $A_j = 3$, there cannot be a 1 in the row where there is a 1 contributed by the variable with $A_j = 1$. However, there can be at most one 1 in the same rows where there are 1s contributed by the variable with $A_j = 2$. Therefore, the variable

with $A_j = 3$ can have 2 rows uniquely covered. The minimum number of constraints possible in this case is 5. Similarly, the table also shows the minimum number of constraints for any $k_{\max}$ value, which is the sum of the values in the shaded squares up to that particular column. By induction, $m_{\min} = 2k_{\max} - 1$.

Table 8 Configuration of 1s for $m_{\min}$

$k_{\max}$		Consecutive and unique k														$m_{\min}$
		2	3	4	5	6	7	8	9	10	11	12	13	14	15	
2	1 +	2														3
3		1	2													5
4		1	1	2												7
5		1	1	1	2											9
6		1	1	1	1	2										11
7		1	1	1	1	1	2									13
8		1	1	1	1	1	1	2								15
9		1	1	1	1	1	1	1	2							17
10		1	1	1	1	1	1	1	1	2						19
11		1	1	1	1	1	1	1	1	1	2					21
12		1	1	1	1	1	1	1	1	1	1	2				23
13		1	1	1	1	1	1	1	1	1	1	1	2			25
14		1	1	1	1	1	1	1	1	1	1	1	1	2		27
15		1	1	1	1	1	1	1	1	1	1	1	1	1	2	29

Consecutive Value Case

If $n_k^* \geq 1$ for $1 \leq k \leq k_{\max}$, then for each $n_k^* \geq 1$, there must be at least one row added in the constraint matrix. Consider the examples shown above for the consecutive and

unique case of n_k^*s and calculate the minimum number of constraints in each of the cases when one or more $n_k^* \geq 1$. Recall the case of $k_{\max}$=2 and $n_1^* = n_2^* = 1$:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Now, consider the case for $k_{\max}$=2 and $n_2^* = 2$, i.e., there are two variables with optimal value 1 for which $A_j = 2$. The portion of the constraint matrix for the variables with optimal value 1 would look like

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

The second matrix here is just the first matrix with the second and third rows swapped. The additional variable with $A_j = 2$ has added one more row in the constraint matrix.

What happens if there is a different repetition pattern? Suppose that n_1^*=2 and $n_2^* = 3$. Then the two possible configurations of the five columns corresponding to variables with optimal value 1 are:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

There are many more instances of the matrices shown above that could be realized by swapping two rows or two columns. The minimum number of constraints is greater by the total number of repetitions than it is in the case where there are no repetitions among n_k^* values at all. In the consecutive and unique values of n_k^*s case, for $k_{\max}$=2, $m_{\min} = 2k_{\max} - 1$ only. Since in this consecutive case, there might be some repetitions of n_k^*s, and each repetition should be accounted for by the formula. A similar empirical formula to that derived above could be used with the additional terms to account for the repeated n_k^*s. The modified expression for calculating the minimum number of constraints becomes:

$$m_{\min} = \left(2k_{\max} - 1\right) + \sum_{k:n_k^*>1} \left(n_k^* - 1\right)$$

Some calculations of the minimum numbers of constraints in the examples above using this new formula are shown below.

$k_{\max}$=2 and $n_2^* = 2$

$$m_{\min} = \left(2k_{\max} - 1\right) + \sum_{k:n_k^*>1} \left(n_k^* - 1\right)$$
$$= 2*2 - 1 + (1-1) + (2-1)$$
$$= 4$$

The second example with n_1^*=2, n_2^*=3.

$$m_{\min} = \left(2k_{\max} - 1\right) + \sum_{k:n_k^*>1} \left(n_k^* - 1\right)$$
$$= 2*2 - 1 + (2-1) + (3-1)$$
$$= 6$$

This formula also applies to the cases of consecutive and unique n_k^* values.

General Case

This is the most general case where $n_k^* \geq 1$ for $1 \leq k \leq k_{\max}$. As explained earlier, if every variable with a value 1 covers at least one unique row of the constraint matrix, then SVC is avoided. In the minimum constraint condition for continuous cases, exactly one row is covered by a variable with optimal value 1 except when there is a variable with optimal value 1 and the corresponding $n_{k_{\max}}^* = 1$. In general, there would be a reduction of as many rows as the number of n_k^*=0 such that $1 \leq k \leq k_{\max}$. So, the final formula for all cases could be written as,

$$m_{\min} = (2k_{\max} - 1) + \sum_{k:n_k^*>0} (n_k^* - 1) - \sum_{k:n_k^*=0} [1]$$

The final term in the formula applies to any n_k^*s ($1 \leq k \leq k_{\max}$) that have value zero. For every instance of n_k^*s $(1 \leq k \leq k_{\max}) = 0$, this term adds 1 and ultimately the total number of cases of zero-valued n_k^*s is deducted from the calculation for the consecutive values case. For example, let us consider $k_{\max}$=4, where n_k^*s for k = 1, 3 and 4 have one repetitive instances. The minimum number of constraints in this case is 10. Similarly, applying the formula for the situation, where $k_{\max}$=4, and $n_1^* = 2,\ n_2^* = 0,\ n_3^* = 2,$ and $n_4^* = 2$, the minimum number of constraints is

$$m_{\min} = \left(2k_{\max} - 1\right) + \sum_{k:n_k^* > 0} \left(n_k^* - 1\right) - \sum_{k:n_k^* = 0} [1]$$
$$= (2 * 4 - 1) + \left(2 - 1\right) + \left(1 - 1\right) + \left(2 - 1\right) - \left[1\right]$$
$$= 8$$

Rationale behind Recommended Guidelines

There are several guidelines and recommendations made during the different phases of the problem generation procedure explained in this chapter. This section explains the rationale behind two of the recommended guidelines adopted for the SCP instance generation procedure.

Value for Adjusted Dual Variable

During the dual variable adjustments, ∂ (a small non-negative value) is assigned to the adjusted dual variable so that 1s can be assigned in the column $j \in J \setminus J^*$ such that the dual constraint corresponding to the column being generated is satisfied. There are, in fact, a range of values for ∂ that can be used in the generation procedure. The range of values for ∂ depends on the marginal distributions of objective function coefficients and the column sums and the target population correlation between these coefficients. However, the safest value can be determined by considering the worst case scenario.

The safest value for ∂ is the value which, when assigned to any non-negative

dual variable during the adjustment procedure, ensures that the row corresponding to the adjusted dual variable can be assigned 1s in a column pertaining to columns $j \in J \setminus J^*$. For any column $j \in J \setminus J^*$, the worst case will be the case when the corresponding c_j is the smallest value and the A_j is the highest among all A_j s. Generally, this case is more likely while generating coefficients from the minimum correlation joint distribution; however, this condition is also possible whenever coefficients are generated under independence based on a composite joint distribution.

Assigning higher values of ∂ may not ensure the adjustment of dual variables will be effective. For example, there might be a case where one or more columns $j \in J \setminus J^*$ have c_j s lower than ∂. If these columns require adjustments of the dual variables and the dual variables are adjusted with the larger ∂, this adjustment would be ineffective for those columns with $c_j < \partial$. Therefore, the safest maximum value of $\partial = \frac{\text{Min } c_j}{\text{Max } A_j}$ and the smallest possible value for ∂ is 0.

Column Generation for Variable with Optimal value 0

It is recommended that for columns $j \in J \setminus J^*$ one should begin in the ascending order of c_j if there are several columns with the same A_j s. For example, if there are five columns such that $j \in J \setminus J^*$ with A_j =*k*, then the column generation should begin from

the smallest c_j to the largest c_j. This rule is not mandatory; however, it eliminates the possibility of running out of the possible configurations of 1s for some column $j \in J \setminus J^*$ despite dual variables assignments that ensure there exist enough configurations of 1s for the column sums generated.

For example, suppose x_1 and x_2 have optimal value 0 with $A_j = 3$ and their c_j s are 7 and 42, respectively. The dual variables corresponding to 6 primal constraints are 0, 0, 6, 10, 12 and 13. After the assignment of dual variables it was determined that x_1 can have one possible configuration of 1s while x_2 can have 20 different configurations of 1s. In this case, if the suggestion to generate the column with the smallest c_j is not adopted and x_2 is generated first, then there is a 1/20 chance that x_2 will have its 1s assigned in the rows with corresponding dual variables as 0, 0 and 6. This is the only configuration that x_1 can assume, and this assignment will make the column, pertaining to x_1 impossible to be generated.

This kind of situation where a unique column satisfying its dual constraint is impossible to generate may arise even when there are several columns with the same A_j. Upon randomly generating columns with larger c_j s, it may so happen that the valid configurations of 1s for one or several columns with smaller c_j are taken by columns with larger c_j s thereby making columns with smaller c_j s impossible to generate with valid configurations of 1s.

In the next chapter, a computational demonstration of this procedure for

generating SCP instances with known optimal solution and correlated coefficients is summarized and analyzed.

CHAPTER FOUR: COMPUTATIONAL STUDIES AND FINDINGS

A computer software program was developed for the SCP generation procedure using the MATLAB® language. A total of 525 SCP instances were generated and solved using three simple greedy heuristics. This chapter summarizes the observations made during the generation process of SCP instances. This chapter also explains the experimental setup, observations made about the SCP instances generated, and computational results for the quality of the solutions found by the three SCP heuristics.

Experimental Setup and Preparation

The SCP instances were created with 100 variables each. The probability that a decision variable assumes value 1 in the optimal solution was set to 0.15 for each variable. Two factors were controlled for the purpose of generating SCP instances: the number of constraint and the population correlation between the objective function coefficients and the column sums of the constraints matrix. A total of seven target population correlation levels and three levels for the number of constraints were considered. For each combination of factor levels, a total of 25 SCP instances were generated. The number of decision variables with optimal value 1 is kept constant for the

corresponding instances across all factor level combinations. For example, the number of variables with optimal value 1 in the first instance for each of the 21 factor level combinations is the same. This is also the case for the second instances for each of the 21 factor level combinations, and so on. Table 9 shows the two factors and their levels with the notations that will be used in summarizing the findings.

Table 9 Experimental factor and their levels

Constraints \ Groups		Correlation levels						
		G1	G2	G3	G4	G5	G6	G7
Actual number	Level	ρ^-	-⅔ ρ^-	-⅓ ρ^-	0	⅓ ρ^+	⅔ ρ^+	ρ^+
$\lceil 0.8m_{\min} + 0.2m_{\max} \rceil$	R1							
$\lceil 0.5m_{\min} + 0.5m_{\max} \rceil$	R2							
$\lceil 0.2m_{\min} + 0.8m_{\max} \rceil$	R3							

SCP Coefficient Generation

For computational demonstration purposes, the following distributions were chosen for the coefficients' values. The objective function coefficients are distributed as $C_j \sim U\{24, 25, ..., 176\}$, and the column sums of the constraint matrix are distributed as $A_j \sim U\{2, 3, ..., 10\}$. Therefore, $\mathrm{E}(C_j) = 100$ and $\mathrm{E}(A_j) = 6$. Since there are 153 values for

the objective function coefficients and nine possible values for the column sums of the constraint matrix, the maximum possible population correlation with these distributions is

$$\rho^{+} = \frac{153}{9}\cdot\sqrt{\frac{9^2-1}{153^2-1}} = 0.993829.$$

It follows that the minimum possible population correlation is $\rho^{-} = -0.993829$.

As shown in Table 9, for each correlation level, a total of 75 SCP instances (25 instances for each of 3 levels for the number of constraints) are needed. Since each instance has 100 variables, a total of 7500 pairs of coefficients are needed for each correlation level. For each decision variable a set of three random numbers are used to generate correlated coefficients by the ECI method as explained in Chapter 2. This was achieved by generating 21 streams of random numbers of length 7500 using MATLAB®. The composite distribution

$$\lambda_0 f_A(a) f_C(c) + \lambda_1 g^{-}(a,c) + \lambda_2 g^{+}(a,c)$$

is used to induce correlation among the SCP coefficients. For any correlation ρ such that $\rho^{-} < \rho < \rho^{+}$, there are an infinite number of such composite distributions. To identify the composite distributions used for this demonstration, a scheme suggested by Reilly (2006b) is used. Specifically, the weights for the composite distributions are:

$$\lambda_0 = \psi(1 - |\rho| / \rho^{+})$$
$$\lambda_1 = \left(1 + \psi(|\rho| / \rho^{+} - 1) - \rho / \rho^{+}\right) / 2$$
$$\lambda_2 = \left(1 + \psi(|\rho| / \rho^{+} - 1) + \rho / \rho^{+}\right) / 2$$

where $0 \le \psi \le 1$. In this demonstration, $\psi = 0.5$ was used consistently so that a median level of independent sampling is expected in the SCP instances simulated for each correlation level. Table 10 shows the different composition weights $(\lambda_0, \lambda_1, \lambda_2)$ used for each of the target correlation levels (ρ).

Table 10 ECI parameters used for SCP generation

Correlation	G1	G2	G3	G4	G5	G6	G7
ρ	-0.99383	-0.6626	-0.3313	0	0.33128	0.66255	0.99383
λ_0	0	0.16667	0.33333	0.5	0.33333	0.16667	0
λ_1	1	0.75	0.5	0.25	0.16667	0.08333	0
λ_2	0	0.08333	0.16667	0.25	0.5	0.75	1

Table 11 shows some descriptive statistics for the sample correlations induced among the SCP coefficients for each target population correlation level.

Table 11 Summary of sample coefficient correlations in the SCP instances generated

	G1	G2	G3	G4	G5	G6	G7
Target correlation	-0.9938	-0.6626	-0.3313	0	0.33128	0.66255	0.99383
Mean	-0.99383	-0.66854	-0.33505	0.01863	0.30993	0.64663	0.99392
Std. deviation	0.00088	0.08347	0.12573	0.10140	0.13802	0.09693	0.00078
Max	-0.99053	-0.49768	0.06024	0.32531	0.58332	0.84689	0.99547
Min	-0.99536	-0.82875	-0.53159	-0.18532	-0.04048	0.30230	0.99169
Range	0.00483	0.33107	0.59183	0.51063	0.62380	0.54459	0.00378

Note that the means of the sample correlations are normally close to the target

correlations. The variability in the sample correlations increases generally as the absolute value of the target correlation decreases. This is attributed to the proportion of coefficient pairs generated under independent sampling.

J^* and m in the Generated SCP Instances

The number of variables with optimal value 1 is kept constant for the same SCP instance (first, second, etc.) across all factor level combinations. Table 12 shows the number of variables with optimal value 1 for the SCP instances generated for every one of the 21 factor combinations.

Table 12 Number of variables with optimal value 1

Problem Instances	$\lvert J^* \rvert$				
1 to 5	12	18	16	15	15
6 to 10	14	14	14	16	17
11 to 15	18	15	12	17	18
16 to 20	14	17	14	18	16
21 to 25	14	14	17	13	13

The number of constraints for each SCP instance depends on the values of the coefficients generated and the level of the number of constraints, R1, R2 or R3. Table 13 shows the summary statistics for the number of constraints generated for each factor level combination.

Table 13 Summary Statistics for number of constraints for SCP instances

Correlation level	Constraints Level R1				Constraints Level R2				Constraints Level R3			
	Mean	Max	Min	Range	Mean	Max	Min	Range	Mean	Max	Min	Range
G1	38.5	47	31	16	58.7	73	45	28	81.7	106	62	44
G2	37.9	46	29	17	59.2	74	39	35	81.2	105	62	43
G3	37.5	45	29	16	56.2	70	43	27	79.2	109	59	50
G4	37.5	45	30	15	59.8	77	47	30	80.3	102	59	43
G5	37.1	46	26	20	58.0	78	39	39	78.6	101	64	37
G6	36.7	44	30	14	55.7	68	40	28	78.7	104	58	46
G7	37.3	46	29	17	55.7	68	40	28	79.7	98	56	42

Other Observations Made during SCP Instances Generation

It was observed that only 30 instances out of 525 required two attempts or more to generate one or more J^* columns. The average of the maximum numbers of attempts for a single column for these 30 instances was 6.56. Similarly, 22 out of 525 required more than 1 attempt to generate one or more $J \setminus J^*$ columns. The maximum number of attempts to generate a $J \setminus J^*$ column for these 22 instances was 2.

Out of 525 instances of SCP generated, 56 instances required the execution of the row sum adjustment procedure. Out of these 56 instances, 39 instances were successfully adjusted by row sum adjustment Procedure 1 while the remaining 16 instances were adjusted by the row sum adjustment Procedure 2. Only in one case did both the procedures fail to adjust the row sums, and hence, that instance was regenerated.

Computational Experiments and Findings

The SCP instances generated were solved using three greedy heuristics: the Drop Heuristic (DH), the Add Heuristic (AH), and the Add/Drop Heuristic (ADH). The following sections briefly explain the procedure for these heuristics.

Drop Heuristic (DH) or Primal Heuristic

This heuristic assumes all variables have value 1 initially. In every iteration, the variable with the highest cost and which would not violate primal feasibility if it were set to 0 is deselected, or "dropped". The procedure terminates when no more variables may be dropped without violating primal feasibility. Since this procedure begins with a feasible solution, DH may be considered a primal-based procedure.

Add Heuristic (AH) or Dual Heuristic

This heuristic adds a variable with value one to a partial solution. The variable selected is the one for which the cost per new row covered is the minimum. This process is repeated until primal feasibility is achieved, i.e, until all rows are covered. Since the procedure begins without having an established feasible solution, AH may be considered a dual-based procedure.

Add/Drop Heuristic (ADH) or Dual/Primal Tandem Heuristic

ADH is the tandem combination of the two heuristics AH and DH mentioned above. The final solution obtained from AH is checked by DH for any possible variables that can be dropped. ADH terminates when none of the variables in the current solution can be dropped without causing infeasibility. This heuristic is generally better than the two heuristics mentioned above. Clearly, it can never be worse than AH.

Two primary statistics were collected for each heuristic solution found: the solution value and the corresponding solution vector. Three secondary statistics were then derived from the primary data to measure the quality of the solutions found by the three greedy heuristics:

1. Relative error: This is the ratio of the heuristic solution value divided by the optimal value, minus 1. Mathematically, relative error $= \frac{Z_{heur}}{Z^*} - 1$, where Z_{heur} is the solution value associated with the heuristic solution and Z^* is the optimal solution value.
2. Optimality: This is the Boolean value (0 (no) or 1 (yes)) indicating whether the heuristic solution is also an optimal solution.
3. Number of discrepancies: These are the counts of differences in the values of the decision variables in the solution vector given by a heuristic compared to the values of the same decision variable in the known optimal solution vector.

Relative Error

Table 14 shows the average relative errors for all three greedy heuristics across all target population correlation levels. It shows that the relative error tends to increase with the increase in population correlation among the coefficients and clearly it is the highest for the maximum correlation condition. Additionally, Table 14 suggests that, among the three greedy heuristics, ADH is better than AH and AH is better than DH. This observation is also evident from the graphical presentation in Figure 14.

With the summary statistics shown in Table 14 and Figure 14, it is quite evident that the population correlation does matter as far as the quality of the solution given by the heuristics are concerned. These results do not consider the effect of the number of constraints on the quality of the heuristic solution found for the SCP instances generated. How the number of constraints affects the quality of the solution for individual heuristics is explained in the following sections.

Table 14 Average relative error for each correlation level for individual heuristics

Correlation	DH	AH	ADH
G1	10.16%	6.22%	1.67%
G2	13.28%	7.35%	2.50%
G3	14.54%	6.12%	1.87%
G4	16.48%	8.37%	3.05%
G5	17.91%	9.48%	3.97%
G6	19.28%	10.90%	5.12%
G7	22.11%	20.39%	11.01%

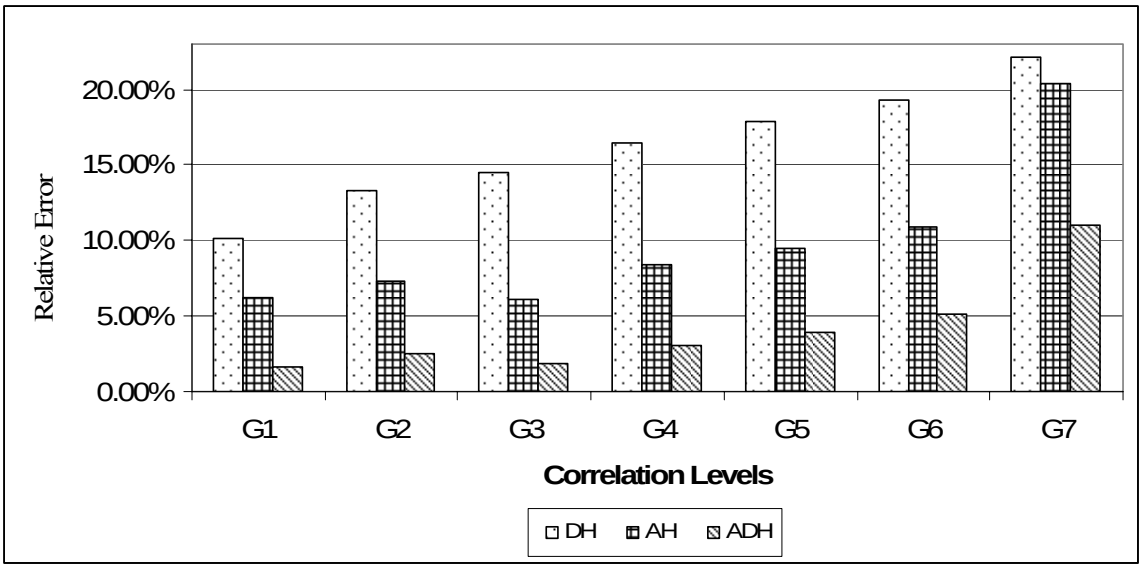


Figure 14: Average relative error for different correlation levels

Drop Heuristic

Summaries of the average relative errors for DH are shown in Table 15 and Figure 15.

Table 15 Average relative errors for Drop Heuristic

Correlation	Constraints Level			Grand Average
	R1	R2	R3	
G1	0.00%	8.88%	21.60%	10.16%
G2	2.99%	11.64%	25.22%	13.28%
G3	5.29%	11.92%	26.43%	14.54%
G4	7.13%	12.55%	29.76%	16.48%
G5	7.65%	14.45%	31.61%	17.91%
G6	9.05%	17.32%	31.46%	19.28%
G7	9.55%	21.16%	35.61%	22.11%

It is observed that, for each level of the number of constraints, the relative error gradually increases with the population correlation levels. It was also observed that the average relative error for the SCP instances with a high number of constraints (level R3) is always larger than that of the other two levels of the number of constraints. Constraints level (R1) has the lowest relative error across all correlation levels. DH performs consistently worse than the other two greedy heuristics, and it performs worst when the correlation level and the number of constraints are at the maximum levels.

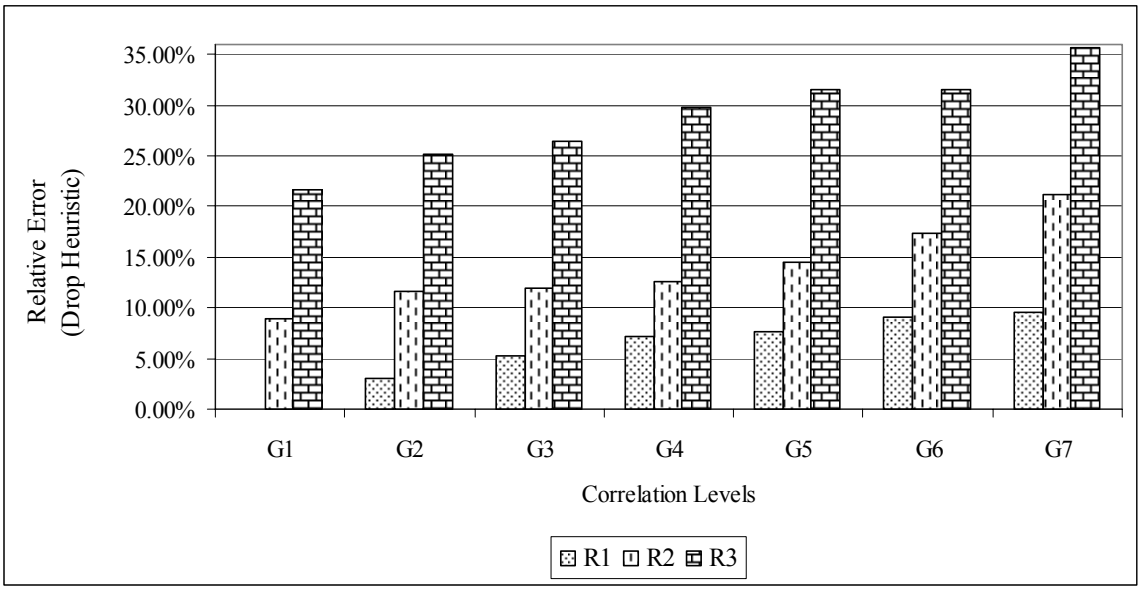


Figure 15: Average relative error for Drop Heuristic

Add Heuristic

Summaries of the average relative errors for AH are shown in Table 16 and Figure 16. The average relative errors increase with the increase in the number of constraints across all correlation levels; however, the differences in average relative errors for constraints level R3 and constraints level R2 with the same correlation level are not as great as in the case of DH, except for the correlation level G7. The average relative errors for constraints level R3 are not consistently greater than the average relative errors for constraints level R2 as seen in the case of DH. Instead, most of the differences in the average relative errors are less than 1% for constraints levels R2 and R3. The slight apparent improvement in performance for correlation level G3 is likely due to natural variation (and to, a lesser extent, the way the number of variables with optimal value 1 was held constant for the first, second, etc instances across all factor level combinations).

Table 16 Average relative errors for Add Heuristic

Correlation	Constraints Level			Grand Average
	R1	R2	R3	
G1	4.66%	7.07%	6.92%	6.22%
G2	5.12%	8.36%	8.56%	7.35%
G3	4.87%	7.50%	5.98%	6.12%
G4	5.64%	9.68%	9.80%	8.37%
G5	6.61%	11.77%	10.06%	9.48%
G6	7.21%	13.19%	12.30%	10.90%
G7	14.56%	20.39%	26.23%	20.39%

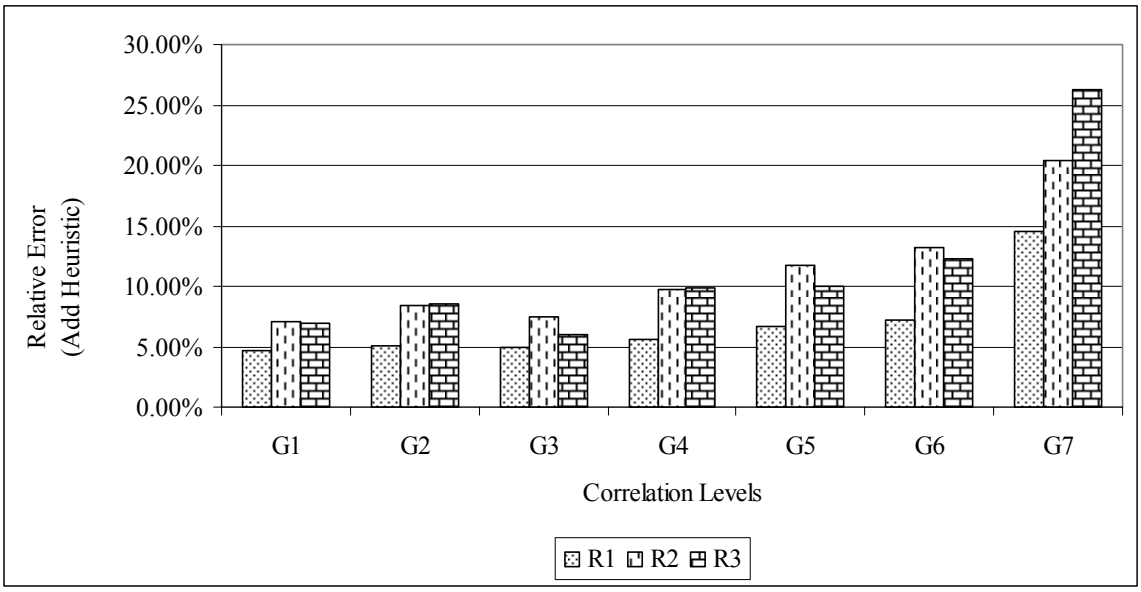


Figure 16: Average relative error for Add Heuristic

Add/Drop Heuristic

Summaries of the average relative errors for ADH are shown in Table 17 and Figure 17.

Table 17 Average relative errors for Add/Drop Heuristic

Correlation	Constraints Level			Grand Average
	R1	R2	R3	
G1	0.29%	2.16%	2.57%	1.67%
G2	0.46%	3.72%	3.31%	2.50%
G3	0.17%	3.79%	1.66%	1.87%
G4	0.68%	3.43%	5.05%	3.05%
G5	1.02%	5.84%	5.05%	3.97%
G6	1.16%	7.46%	6.75%	5.12%
G7	2.46%	9.69%	20.87%	11.01%

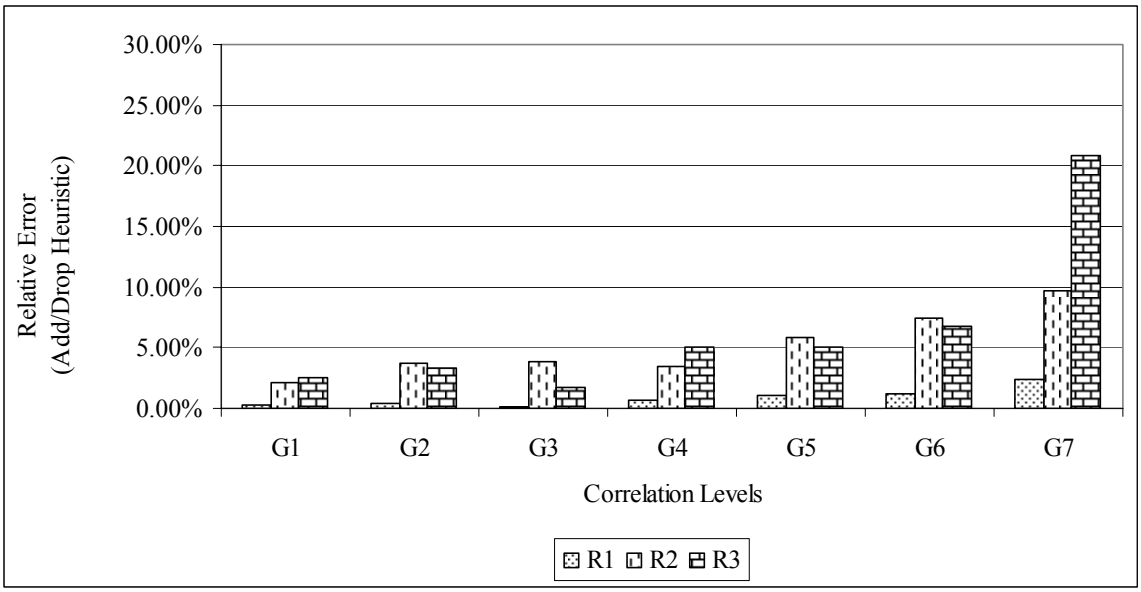


Figure 17: Average relative error for Add/Drop Heuristic

The effect of the number of constraints is very evident for the correlation level G7. The average relative error for constraints level R3 is slightly smaller than the average relative error for constraints level R2 for correlation levels G2, G3, G5 and G6. However, the average relative errors for the rest of the correlation levels are consistent with those of the other two heuristics. This heuristic is the best of the three heuristics considered here in terms of the average relative error. ADH has shown average relative errors less than 8% for all correlation levels except G7, much better than what was observed for the other two heuristics.

Optimality

For each correlation and number of constraints combination, counts of the number of times optimality was achieved were recorded for each heuristic. Table 18 shows the summary of the number of times each of the three heuristics achieved optimality.

AH, which performed better in the case of the average relative error was found to be the worst of all when it comes to the number of times it found the optimal solution value. Out of 525 SCP instances, AH found the optimal value only twice.

DH, which was the worst heuristic as far as relative error is concerned, performed slightly better as it found optimal solutions for all 25 SCP instances for the minimum population correlation level and for one instance with correlation level G2 with the constraints level R1.

ADH found a solution with the optimal value 147 times. Most interestingly, it found the optimal solution value more often for constraints levels R1 and R3 than for constraints level R2 across all correlation levels. The cause of this phenomenon should be determined through additional experimentations.

Table 18 Counts of optimality achieved for all heuristics

Heuristic	Correlation Level	Constraints Level			Grand Total
		R1	R2	R3	
DH	G1	25	0	0	25
	G2	1	0	0	1
	G3	0	0	0	0
	G4	0	0	0	0
	G5	0	0	0	0
	G6	0	0	0	0
	G7	0	0	0	0
DH Total		**26**	**0**	**0**	**26**
AH	G1	1	0	0	1
	G2	0	0	0	0
	G3	1	0	0	1
	G4	0	0	0	0
	G5	0	0	0	0
	G6	0	0	0	0
	G7	0	0	0	0
AH Total		**2**	**0**	**0**	**2**
ADH	G1	13	1	18	32
	G2	10	1	16	27
	G3	13	1	17	31
	G4	5	1	14	20
	G5	6	1	16	23
	G6	4	0	9	13
	G7	1	0	0	1
ADH Total		**52**	**5**	**90**	**147**

Number of discrepancies

Table 19 summarizes the average number of discrepancies for all the heuristics for all correlation levels. Figure 18 shows the same results in graphical form. The average numbers of discrepancies between the heuristic solution vector and the optimal solution vector was highest for DH and they increase as the correlation level increases. The same trend was evident for AH and ADH, except for the correlation level G3, where a slight dip in the average number of discrepancies is observed.

Table 19 Average number of discrepancies for the heuristics

Correlation	DH	AH	ADH
G1	14.51	5.92	3.72
G2	18.68	7.37	5.00
G3	20.43	6.64	4.49
G4	21.39	7.95	5.29
G5	22.11	8.67	6.09
G6	22.69	9.49	7.07
G7	23.29	17.72	12.83

Figure 19 shows the total number of discrepancies for all three heuristics across all combinations of correlation levels and number of constraints levels. It is evident from Figure 19 that the average number of discrepancies almost always increases with the correlation levels and peaks at G7 correlation level for each level of number of constraints for all the heuristics.

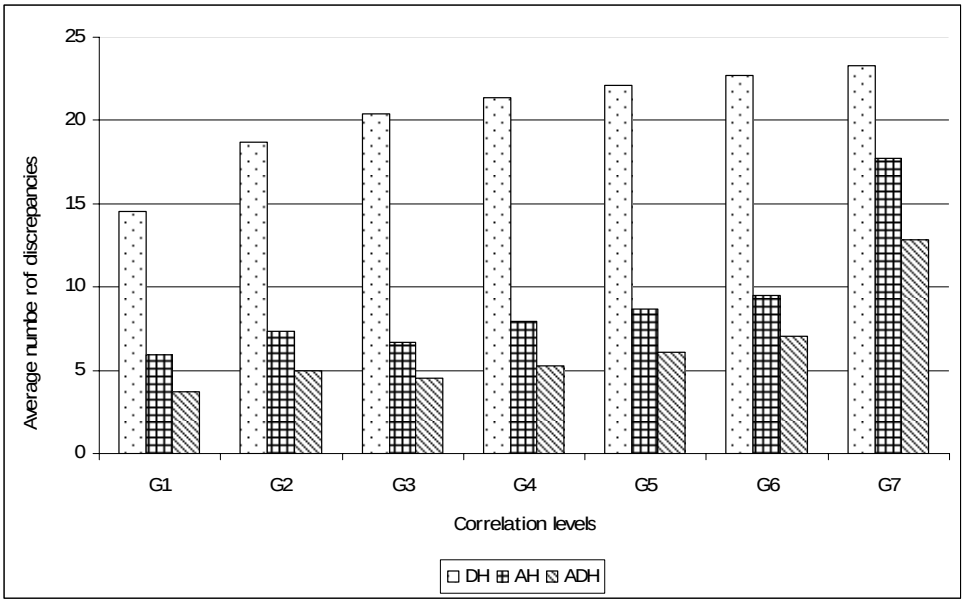


Figure 18: Average number of discrepancies in solution vectors

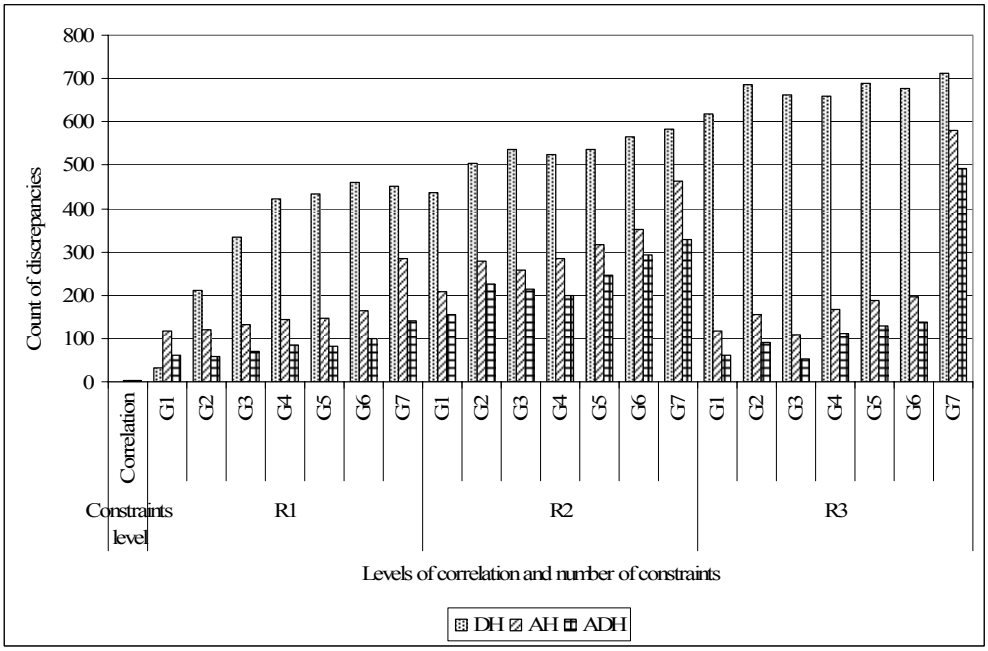


Figure 19: Count of discrepancies in solution vectors

The results and findings presented in this chapter illustrate that the population correlation level between the objective function coefficients and the column sums of the constraints matrix and the level of the number of constraints affect the relative error of the heuristic solutions, the number of times an optimal solution is found by the heuristics and the discrepancies between the heuristic and the optimal solution vectors. ADH is the best performing heuristic overall. Relative errors for AH are smaller than those for DH; however, DH finds more optimal solutions than AH, especially for correlation level G1.

The results for the three heuristics show that fewer optimal solutions and lower-quality heuristic solutions are found as the correlation between SCP coefficients increases. Additionally, it appears more likely that an optimal solution will be found if there are relatively few or relatively many constraints, especially for ADH.

Table 20 summarizes the number of alternate optimal solutions found by the heuristics. Out of 1575 heuristic solutions (525 solutions x 3 heuristics), 35 solutions found by the heuristics are alternate optimal solutions. DH and AH never found alternate optimal solutions except for correlation level G1 and constraint level R1. DH found 12 alternate optimal solutions and AH found only one alternate optimal solution; in all cases, the alternate optimal solutions were found for G1 and R1 combinations. ADH found alternate optimal solutions up to correlation level G6 for constraint level R1.One instance of alternate optimal solutions was found for each of the G3-R3 and G5-R2 combinations. Table 20 also shows the clear trend that the number of alternate optimal solutions

decreases as the coefficients correlation increases or the number of constraints increases.

Table 20 Summary of alternate optimal solutions found by SCP heuristics

Heuristic	Correlation Level	Constraint Levels			Grand Total
		R1	R2	R3	
DH	G1	12	0	0	**12**
	G2	0	0	0	**0**
	G3	0	0	0	**0**
	G4	0	0	0	**0**
	G5	0	0	0	**0**
	G6	0	0	0	**0**
	G7	0	0	0	**0**
AH Total		**12**	**0**	**0**	**12**
AH	G1	1	0	0	**1**
	G2	0	0	0	**0**
	G3	0	0	0	**0**
	G4	0	0	0	**0**
	G5	0	0	0	**0**
	G6	0	0	0	**0**
	G7	0	0	0	**0**
DH Total		**1**	**0**	**0**	**1**
ADH	G1	6	0	0	**6**
	G2	4	0	0	**4**
	G3	4	0	1	**5**
	G4	2	0	0	**2**
	G5	2	1	0	**3**
	G6	2	0	0	**2**
	G7	0	0	0	**0**
ADH Total		**20**	**1**	**1**	**22**

In the next chapter, conclusions drawn from the research, as well as possible extensions for future research in this area, are reported and discussed.

CHAPTER FIVE: CONCLUSIONS

This chapter summarizes the objective and findings of this research. It also outlines some recommendations regarding future research that might extend this work.

This study clearly shows that SCP instances with known optimal solution and with induced target population correlation between objective function coefficients and the column sums of constraint coefficients can be simulated. In order to simulate SCP instances with known optima and specified coefficient correlation, the usual problem generation process must be modified significantly.

This research clearly shows that correlation does matter as far as the quality of the solutions found by heuristics for SCP is concerned. In this study, three greedy heuristics, AH, DH, and ADH, were used to solve the simulated SCP instances for seven different levels of population correlation ranging from the minimum to the maximum correlation possible. The computational results presented in Chapter 4 show that the relative errors of the solutions provided by the heuristics increase with increases in the population correlation between objective function coefficients and the column sums of the constraints matrix.

This study also shows that the likelihood of finding a non-optimal solution increases with the level of coefficient correlation. Out of the three heuristics used in the

study, even the best among the three, ADH, struggles to find the optimal solution when the target population correlation increases to the maximum.

Another important finding of this study is that the number of constraints also affects the quality of the solutions found by the heuristics. For each of the heuristics, the relative error typically increases as the numbers of constraints increases for the same correlation level. So it is fairly safe to conclude that the SCP instances with higher population correlation between the coefficients and a higher number of constraints present greater challenges for SCP heuristics.

Another unexpected finding of this research is that the range for the number of constraints for simulated SCP instances with known optimal solutions depends on the value of the constraint column sums and the known optimal solution. Whenever, a certain number of variables have been selected as variables with optimal value 1, the range for the feasible number of constraints in an SCP instance that can be generated with the coefficients in hand is restricted. The feasible range for the number of structural constraints m , $m_{\min} \le m \le m_{\max}$ can be calculated using the following formulas:

$$m_{\min} = (2k_{\max} - 1) + \sum_{k:n_k^* > 0} (n_k^* - 1) - \sum_{k:n_k^* = 0} [1]$$

$$m_{\max} = \sum_{j \in J^*} A_j$$

where $k_{\max} = \max_{j \in J^*} \{A_j\}$ and $k_{\min} = \min_{j \in J^*} \{A_j\}$. The justification for these formulas is given in the Discussion section in Chapter 3.

Future Work

This endeavor has added knowledge to the existing body of knowledge in the field of optimization, particularly in the area of generating random problems with known optimal solutions. It is also hoped that this will help researchers to develop new methodologies to generate random problems with known optimal solutions for other classes of optimization problems. This research has opened new research opportunities. Outlined below are some of the areas which could be extensions of this research.

1. It is suggested to investigate the performance of SCP heuristics on SCP instances generated with different marginal distributions for the coefficients. Distributions could be of the same family with different parameters, or they could be of different family of distributions or combinations of both.
2. The effect of ψ, a measure of the relative proportion of independent sampling, on the quality of the solutions found by SCP heuristics could be investigated. In this research, ψ=0.5 is used to generate objective function coefficients and column sums of the constraints matrix. It is suggested to investigate the performance of SCP heuristics for different values of ψ as it can assume any real value between 0 and 1.
3. In this research, three greedy heuristics for SCP are used to demonstrate the efficacy of the procedure to generate SCP instances with known optimal solution and explicitly induced correlation between objective function coefficients and

column sums of the constraints matrix. It is suggested to investigate the performance of other SCP heuristics on the quality of the solution. Additionally, the relationship between the number of constraints and the number of optimal solutions found with ADH merits further investigation.

4. This research has shown that SCP instances with known optimal solutions and specified coefficients correlation can be simulated. It is suggested that the lessons learned from this could be extended to other class of optimization problems to randomly generate test problems with known optimal solution and correlated coefficients. Because of the long standing interest in KP01, it is recommended that a procedure for simulating correlated KP01 instances with known optimal solution be developed.

APPENDIX A:
AN EXAMPLE OF SCP GENERATION

An example is shown to demonstrate how a 25-variable SCP instance with a predetermined solution and correlated coefficients is generated. The distributions selected for the cost coefficients (C) and column sums (A) are:

$f_C(c)$ = Uniform {1, 2, ..., 200} is the marginal distribution of cost coefficients, and

$f_A(a)$ = U+V is the marginal distribution of column sums of the constraints matrix, where U ~ Uniform {1, 2, 3} and V~ Uniform {1, 2, 3}.

The target population correlation chosen was 0.5. The composite distribution $\lambda_0 f_A(a) f_C(c) + \lambda_1 g^-(a,c) + \lambda_2 g^+(a,c)$ was used to generate coefficients under ECI where $f_A(a) f_C(c)$ is the joint distribution under independence, $g^-(a,c)$ is the minimum correlation distribution for (A, C), and $g^+(a,c)$ is the maximum correlation distribution for (A, C).

Given the marginal distributions of the coefficient values and the target population correlation, λ_0 = 0.084608, λ_1 = 0.207696, and λ_2 = 0.707696 were selected to characterize the composite distribution of (A, C). 25 values of (A, C) were generated and a solution vector was selected. The number of constraints was chosen to be 20.

The following 6 tables progressively show the generation of an SCP instance through different phases of the generation process, including:

- the generation of "unique" ones in the constraint matrix (Table A.1),
- the generation of the rest of the entries in the columns of the variable with optimal value 1 (Table A.2),
- the initial arrangement of dual variables (Table A.3),

- the adjustment of dual variable values (Table A.4),
- the generation of coefficients in the remaining columns (Table A.5), and
- the adjustment of row sums (Table A.6).

The sample correlation for the coefficients generated is 0.422.

Table A. 1 Generation if I_u

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	
c_j	62	163	13	187	16	23	18	53	61	101	86	182	199	46	90	179	81	148	163	143	95	193	110	11	66	
A_j	3	3	4	6	6	2	6	3	3	4	4	6	6	3	4	6	4	5	5	5	4	6	4	6	3	
x_j	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
i																										s_i
1																										1
2				1																						1
3																										0
4																										0
5									1																	1
6																									1	1
7																										0
8						1																				1
9											1															1
10		1																								1
11																										1
12																										0
13																										0
14																										0
15																										0
16																										0
17																										0
18																										0
19																										0
20																										0

Table A.1 shows that the rows 2, 5, 6, 8, 9 and 10 are now covered uniquely by x_4, x_9, x_{25}, x_6, x_{11}, and x_2, respectively.

Table A. 2 Generation of columns for $j \in J^*$

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	
c_j	62	163	13	187	16	23	18	53	61	101	86	182	199	46	90	179	81	148	163	143	95	193	110	11	66	
A_j	3	3	4	6	6	2	6	3	3	4	4	6	6	3	4	6	4	5	5	5	4	6	4	6	3	
x_j	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
i																										s_i
1		1		0		0			0		0														0	1
2		0		1		0			0		0														0	1
3		0		1		0			0		0														0	1
4		0		0		0			0		0														1	1
5		0		0		0			1		0														0	1
6		0		0		0			0		0														1	1
7		0		0		0			1		0														0	1
8		0		0		1			0		0														0	1
9		0		0		0			0		1														0	1
10		1		0		0			0		0														0	1
11		1		0		0			0		0														0	1
12		0		1		1			0		0														0	2
13		0		0		0			0		1														0	1
14		0		1		0			0		0														0	1
15		0		1		0			0		0														0	1
16		0		1		0			0		0														0	1
17		0		0		0			0		0														1	1
18		0		0		0			1		0														0	1
19		0		0		0			0		1														0	1
20		0		0		0			0		1														0	1

Table A. 3 Initial dual variable assignments

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25		π_i	$\sum_{r=1}^{i} \pi_{<r>}$	<i>
c_j	62	163	13	187	16	23	18	53	61	101	86	182	199	46	90	179	81	148	163	143	95	193	110	11	66				
A_j	3	3	4	6	6	2	6	3	3	4	4	6	6	3	4	6	4	5	5	5	4	6	4	6	3				
x_j	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1				
i																										s_i			
1		1		0		0			0		0														0	1	54.33	0	1
2		0		1		0			0		0														0	1	37.4	20.33	2
3		0		1		0			0		0														0	1	37.4	40.66	3
4		0		0		0			0		0														1	1	22	61	4
5		0		0		0			1		0														0	1	20.33	82.5	5
6		0		0		0			0		0														1	1	22	104	6
7		0		0		0			1		0														0	1	20.33	125.5	7
8		0		0		1			0		0														0	1	23	147	8
9		0		0		0			0		1														0	1	21.5	169	9
10		1		0		0			0		0														0	1	54.33	191	10
11		1		0		0			0		0														0	1	54.33	213	11
12		0		1		1			0		0														0	2	0	236	12
13		0		0		0			0		1														0	1	21.5	273.4	13
14		0		1		0			0		0														0	1	37.4	310.8	14
15		0		1		0			0		0														0	1	37.4	348.2	15
16		0		1		0			0		0														0	1	37.4	385.6	16
17		0		0		0			0		0														1	1	22	423	17
18		0		0		0			1		0														0	1	20.33	477.33	18
19		0		0		0			0		1														0	1	21.5	531.66	19
20		0		0		0			0		1														0	1	21.5	586	20

Table A. 4 Dual variable adjustments

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25		π_i	$\sum_{r=1}^{i} \pi_{<r>}$	< i >
c_j	62	163	13	187	16	23	18	53	61	101	86	182	199	46	90	179	81	148	163	143	95	193	110	11	66				
A_j	3	3	4	6	6	2	6	3	3	4	4	6	6	3	4	6	4	5	5	5	4	6	4	6	3				
x_j	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1				
i																										s_i			
1		1		0		0			0		0														0	1	54.33	0	1
2		0		1		0			0		0														0	1	93.35	0.1	2
3		0		1		0			0		0														0	1	0.1	0.2	3
4		0		0		0			0		0														1	1	32.95	0.3	4
5		0		0		0			1		0														0	1	30.45	0.4	5
6		0		0		0			0		0														1	1	0.1	0.5	6
7		0		0		0			1		0														0	1	0.1	0.6	7
8		0		0		1			0		0														0	1	23	0.7	8
9		0		0		0			0		1														0	1	42.9	23.7	9
10		1		0		0			0		0														0	1	54.33	54.15	10
11		1		0		0			0		0														0	1	54.33	84.6	11
12		0		1		1			0		0														0	2	0	117.55	12
13		0		0		0			0		1														0	1	0.1	150.5	13
14		0		1		0			0		0														0	1	0.1	193.4	14
15		0		1		0			0		0														0	1	93.35	236.3	15
16		0		1		0			0		0														0	1	0.1	290.63	16
17		0		0		0			0		0														1	1	32.95	344.96	17
18		0		0		0			1		0														0	1	30.45	399.3	18
19		0		0		0			0		1														0	1	42.9	492.65	19
20		0		0		0			0		1														0	1	0.1	586	20

Table A. 5 Generation of columns for $j \in J \setminus J^*$

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25		
c_j	62	163	13	187	16	23	18	53	61	101	86	182	199	46	90	179	81	148	163	143	95	193	110	11	66		
A_j	3	3	4	6	6	2	6	3	3	4	4	6	6	3	4	6	4	5	5	5	4	6	4	6	3		
x_j	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1		
i																										s_i	π_i
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	54.33
2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	3	93.35
3	0	0	1	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	0	9	0.1
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	3	32.95
5	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	30.45
6	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	1	8	0.1
7	0	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	0	9	0.1
8	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	4	23
9	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	4	42.9
10	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	5	54.33
11	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	3	54.33
12	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	8	0
13	1	0	0	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0	0	8	0.1
14	0	0	1	1	1	0	1	0	0	1	0	1	0	1	0	1	1	0	0	1	0	0	0	1	0	11	0.1
15	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	3	93.35
16	1	0	0	1	1	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	1	1	0	1	0	11	0.1
17	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2	32.95
18	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	5	30.45
19	1	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	6	42.9
20	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	6	0.1
$\pi \mathbf{A}_j$	43	163	0.4	187	0.6	23	0.5	43	61	97	86	64	152	23	77	157	31	140	127	108	94	117	94	1	66	6	0.1

Table A. 6 Adjustment of row sums

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25		
c_j	62	163	13	187	16	23	18	53	61	101	86	182	199	46	90	179	81	148	163	143	95	193	110	11	66		
A_j	3	3	4	6	6	2	6	3	3	4	4	6	6	3	4	6	4	5	5	5	4	6	4	6	3		
x_j	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1		
I																										s_i	π_i
1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2	54.33
2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	3	93.35
3	0	0	1	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	0	9	0.1
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	3	32.95
5	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	30.45
6	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	1	8	0.1
7	0	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	0	9	0.1
8	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	4	23
9	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	3	42.9
10	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	5	54.33
11	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	3	54.33
12	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	8	0
13	1	0	0	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0	0	8	0.1
14	0	0	1	1	1	0	1	0	0	1	0	1	0	1	0	1	1	0	0	1	0	0	0	1	0	11	0.1
15	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	3	93.35
16	1	0	0	1	1	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	1	1	0	1	0	11	0.1
17	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2	32.95
18	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	5	30.45
19	1	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	6	42.9
20	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	6	0.1
πA_j	43	163	0.4	187	0.6	23	0.5	43	61	97	86	64	152	23	77	157	31	140	127	108	94	117	94	1	66	6	0.1

APPENDIX B:
Avoiding Superfluous Variable Condition

The superfluous variable condition (SVC) occurs when two or more variables a SCP instance would have the same contributions to feasibility in the structural constraints if selected for the optimal solution. For example, if variables x_1 and x_2 cover exactly the same rows, then one of them is superfluous. It is impractical to consider both variables regardless of whether their costs in the objective function are the same or different.

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

For illustration purposes, a binary matrix **A** with five variables and 6 structural constraints is shown above. Variables x_2 and x_3 have the same contribution to feasibility in the structural constraints. The optimal solution for this problem includes either x_2, x_4 and x_5 or x_3, x_4 and x_5. One of the variables, either x_2 or x_3, need not be considered at all. It is practical to remove the one with the higher cost in this situation. When simulating SCP instances, it is possible to avoid such a situation.

One might argue about the status of x_1 here. The only constraint covered by x_1 is the first constraint, which is also covered by both variables x_4 and x_5 which each have value one in the optimal solution. Since this is a small problem, visual inspection might indicate that x_1 is superfluous, but as the problem size increases, there might be a possibility that the first row is covered by x_1 and the remaining rows now covered by x_4

and x_5 will be covered by some other variables such that the combined cost in the objective function is less than that what it would have been if x_4 and x_5 had been selected.

In the example above, x_1 has column sum equal to 1, however in the generation procedure it is assumed that every variable has corresponding column sums greater than or equal to 2. The example above is for illustration purposes only and the key point here is, as long as each variable covers constraints differently, it can be considered a valid candidate as a decision variable.

SVC for variable with optimal value 1

Since the SCP generation procedure assumes the optimal solution beforehand, it is not sufficient for columns in J^* to have configurations of 1s uniquely different from those of the rest of the columns. As the columns in J^* correspond to decision variables with value 1 in the optimal solution, any $x_j \ni j \in J^*$ whose value is changed to 0 should result in violation of the basic structural constraints of the SCP. Sometimes, while generating columns for $x_j \ni j \in J^*$ the configuration of 1s in those columns forms a structure such that one or more variables representing columns in J^* becomes superfluous and do not appear in the actual optimal solution. This is illustrated with the example given below.

Let us consider that we have five decision variables already selected to have value 1 in the optimal solution. The matrix given below is the portion of the constraint matrix for the columns in J^* only. Close observation of the matrix shows that x_2 is superfluous as the rows 2 and 6 covered by this variable are also covered by x_3 (which covers row 4, 6 and 7) and x_5 (which covers row 1, 2 and 7). We can not remove either x_3 or x_5 as they cover at least one unique row, namely rows 4 and 1. Therefore, x_2 must not be in the solution for this configuration of columns in J^* and, if it is removed, there will not be a violation of the primal constraints. If it is desired to have x_2 as the variable with optimal value 1, then the configurations of 1s should be changed in such a way that x_2 covers as least one constraint row uniquely.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Checking for a valid configuration of 1s for the columns in J^* could become unmanageable as the constraint-matrix size increases. A small checking procedure should be applied to see whether the newly formed configuration is valid. Let $\oplus$ denote an operation among equal size binary vectors such that each element in the resultant vector is the largest value among the corresponding elements of the participating vectors. For

example, consider,

$$\begin{bmatrix}1\\0\\0\\1\end{bmatrix} \oplus \begin{bmatrix}0\\0\\1\\1\end{bmatrix} = \begin{bmatrix}1\\0\\1\\1\end{bmatrix}$$

In the example above, the resulting vector has as its elements the larger of the corresponding elements of the participating vectors. For every column in J^*, a resultant vector of all already filled columns in J^* is calculated excluding the current column in J^* and if the resultant vector has 1s in every place that the current column in J^* in question has then the current column is superfluous. If not, the current column covers at least one unique row.

Remedy for SVC for variable with optimal value 1

The task of keeping track of whether or not a current column generation for a variable with index in J^* has made any of the already generated variables with indices in $J \setminus J^*$ superfluous is difficult. It is because even if a current column $j \in J^*$ is unique from those columns $j \in J^*$ already generated, possibilities may still exist that the insertion of a newly generated column may turn one or more variables corresponding to the columns $j \in J^*$ already generated redundant or superfluous. An example shown below illustrates this fact.

Consider 4 decision variables with value 1 in the optimal solution. Three columns

generated for x_1, x_2 and x_3 are the valid columns as none of them is superfluous. The column corresponding to x_4 is now generated as shown below.

$$\underset{j\in J^*=\{1,3\}}{A} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix},\ x_4 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},\ \underset{j\in J^*}{A} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To check whether x_4 is superfluous or not, the resultant vector of x_1, x_2 and x_3 is created and checked to see if, for every entry of 1 in the column corresponding to x_4, there is a 1 in the corresponding entry in the resultant vector. Since that is not the case here, x_4 is not superfluous. However, it is required to check if any of x_1, x_2 and x_3 has now become superfluous. In this case, x_1 has become superfluous since it does not cover any unique row like the other variables do. The generation of columns $j \in J^*$ thus becomes a cumbersome process. In order to avoid complications arising from the SVC, a simple rule can be established such that the possibility of encountering SVC can be eliminated.

Columns with binary entries can be made linearly independent if there is at least one row for every column $j \in J^*$ that is uniquely covered. Therefore, for every column $j \in J^*$, if one random row is made to be uniquely covered by that column and no 1s are assigned in that row for the remaining columns in J^* then this column is linearly independent from the rest of the columns $j \ni j \in J^*$. This makes at least $|J^*|$ rows

uniquely covered by columns in J^*. The remaining rows, which are not uniquely covered by columns in J^*, can have all the remaining 1s in the columns $j \in J^*$. These remaining rows, which were not uniquely covered, may now become either uniquely covered or may be covered by the combinations of columns in $j \in J^*$. This rule certainly avoids encountering SVC and the tedious process of determining the resultant vector to determine whether the SVC has occurred or not with the newly generated column $j \in J^*$. The SCP generation procedure generates instances such that each variable with value 1 covers a unique row. Therefore, there are no superfluous variables in the SCP instances generated by this procedure.

LIST OF REFERENCES

Arthur, J. L., & Frendewey, J. O. (1988). Generating Traveling-Salesman Problems with Known Optimal Tours. *The Journal of the Operations Research Society*, 39(2), 153-159.

Bard, J. F., & Purnomo, H. W. (2005). Preference scheduling for nurses using column generation. *European Journal of Operations Research*, 164, 510-534.

Beasley, J. E. (1990). A lagrangian heuristic for set-covering problems. *Naval Research Logistics*, 37, 151-164.

Balas, E., & Carrera, M. (1996). A dynamic subgradient-based branch-and-bound for set covering. *Operations Research*, 44(6), 875-890.

Balas, E., & Ho, A. (1980). Set covering algorithms using cutting planes, heuristics and subgradient optimization: A computational study. *Mathematical Programming*, 12, 37-60.

Balas, E., & Padberg, M. W. (1976). Set Partitioning: A Survey. *MSIAM Review*, 18(4), 710-760.

Balas, E., & Zemel, E. (1980). An algorithm for large zero-one knapsack problems. *Operations Research*, 28(5), 1130-1154.

Bard, F. D., & Purnomo, H. W. (2005). Preference scheduling for nurses using column generation. *European Journal of Operations Research*, 164, 510-534.

Brown, G. G., Graves, G. W., & Ronen, D. (1987). Scheduling ocean transportation of crude oil. *Management Science,* 33, 335-346.

Caprara, A., Fischetti, M., & Toth, P. (1999). A heuristic method for the set covering problem. *Operations Research*, 47(5), 730-743.

Caprara, A., Fischetti, M., & Toth, P. (2000). Algorithms for the Set Covering Problems. *Annals of Operations Research*, 98, 353-371.

Cardei, M., & Du, D. (2005) Improving Wireless Sensor Network Lifetime through Power Aware Organization. *Wireless Networks*, 11(3), 333 – 340.

Cario, M., Clifford, J., Hill, R., Yang, J., Yang, K., & Reilly, C. H. (2002). An investigation of the relationship between problem characteristics and algorithm performance: a case study of the GAP. *IIE Transactions*, 34, 297-312.

Ceria, S., Nobili, P., and Sassano, A. (1998). Set covering problem. In M. Dell' Amico, F. Maffioli, & S. Martello (Eds.). Annotated Bibliographies in Combinatorial Optimization.
John Wiley & Sons: UK.

Fisher, M. L., and Rosenwein, M. B. (1989). An interactive optimization system for bulk-cargo ship scheduling. *Naval Research Logistics,* 36, 27-42.

Hill, R. R. (1998). An analytical comparison of optimization problem generation methodologies. In D. J. Medeiros, E. F. Watson, J. S. Carson, & M. S. Manivannan (Eds.), *Proceedings of the 1998 Winter Simulation Conference*, 609-615.

Hill, R., & Reilly, C. H. (1994). Composition of multivariate random variables. . In J. T. Tew, S. Manivannan, R. P. Sadowski, & A. F. Seila (Eds.), *Proceedings of the 1994 Winter Simulation Conference*, 332-342.

Hill, R., & Reilly, C. H. (2000a). Multivariate composite distributions for coefficients in synthetic optimization problems. *European Journal of Operations Research*, 121, 64-71.

Hill, R., & Reilly, C. H. (2000b). The effects of coefficient correlation in two-dimensional knapsack problems on solution procedure performance. *Management Science,* 46(2), 302-317.

Hooker, J. (1994). Needed: an empirical science of algorithms. *Operations Research*, 42(2), 201-212.

Jacobs, L. W., & Brusco, M. J. (1995). Note: A local-search heuristic for large set-covering problems. *Naval Research Logistics*, 42, 1129-1140.

John, T. (1989). Tradeoff solutions in single machine production scheduling for minimizing flow time and maximum penalty. *Computers and Operations Research*, 16(5), 471-479.

Karp, M. P. (1972). Reducibility among combinatorial problems. In R. E. Miller, & J. W. Thatcher (Eds.), Complexity of Computer Computations. New York: Plenum Press.

Martello, S., Pisinger, D., & Toth, P. (1999). Dynamic programming and strong bounds for 0-1 knapsack problems. *Management Science*, 45(3), 414-424.

Martello, S., Pisinger, D., & Toth, P. (2000). New trends in exact algorithms for the 0-1 knapsack problems. *European Journal of Operations Research*, 123, 325-332.

Martello, S., & Toth, P. (1979). The 0-1 knapsack problem. In A. Mingozzi & C. Sandi (Eds.), Combinatorial Optimization. New York: John Wiley and Sons.

Martello, S., & Toth, P. (1997). Upper bounds and algorithms for 0-1 knapsack problems. *Operations Research*, 45(5), 64-71.

Moore, B. A., Peterson, J. A., & Reilly, C. H. (1990). Characterizing distributions of discrete bivariate random variables for simulation and evaluation of solution methods. In O. Balci, R. P. Sadowski, & R. E. Nance (Eds.), *Proceedings of the 1990 Winter Simulation Conference*, 294-302.

Nelson, R. B. (1987). Discrete bivariate distributions with given marginals and correlation. *Communications in Statistic: Simulation and Computations,* 16(1), 199-208.

O'Neill, R. P. (1982). A Comparison of Real-World Liner Programs and their Randomly Generated Analogs. In J. M. Mulvey (Ed.), Evaluating Mathematical Programming Techniques, Lecture Notes in Economics and Mathematical Systems No 199. Berlin: Springer-Verlag.

Pilcher, M. G. & Rardin, R. L. (1992). Partial polyhydral description and generation of discrete optimization problems with known optima. *Naval Research Logistics*, 39, 839-858.

Pisinger, D. (1997). A minimal algorithm for the 0-1 new trends in exact algorithms for the 0-1 knapsack problem. *Operations Research*, 45(5), 758-767.

Potts, C. & Wassenhove, L. V. (1988). Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Science*, 34(7), 843-858.

Rardin, R. L. & Lin, B. W. (1982). Test problems for computational experiments- issues and techniques. In J. M. Mulvey (Ed.), Evaluating Mathematical Programming Techniques. Berlin: Springer-Verlag.

Reilly, C. H. (1991). Optimization test problems with uniformly distributed coefficients. In B. L. Nelson, W. D. Kelton, & G. M. Clark (Eds.), *Proceedings of the 1991 Winter Simulation Conference*, 866-874.

Reilly, C. H. (1997). Generating coefficients for optimization problems with implicit correlation induction. *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, 3, 2438-2443.

Reilly, C. H. (1999). Input models for synthetic optimization problems. In P. A. Farrington, H. B. Nembhard, D. T. Sturrock, & G. W. Evans (Eds.), *Proceedings of the 1999 Winter Simulation Conference,* 116-121.

Reilly, C. H. (2006a). Synthetic optimization-problem generation: Show us correlations! Unpublished manuscript, University of Central Florida, Orlando, FL.

Reilly, C. H. (2006b). Coefficients Ratios in Simulated 0-1 Knapsack Problems. In N. Callaos, D. Zinn, M. J. Savoie, X. Hu, R. Hill, & H. Haga (Eds.), *Proceedings of The 10th World Multi-Conference on Systemics, Cybernetics and Informatics*, 6, 51-56.

Rushmeier, R. & Nemhauser, G. (1993). Experiments with parallel branch-and-bound algorithms for the set covering problem. *Operations Research Letters*, 13(5), 277-285.

Smith B. M., & Wren A.(1988). A Bus Crew Scheduling System Using a Set Covering Formulation. , *Transportation Research,* 22(A), 97-108.

Vasko, F.J., Wolf, F.E., and Stott, K.L. (1987). Optimal selection of ingot sizes via set covering, *Operations Research,* 35, 346-353.