Finding Improving Solutions that Control Disruption to Binary Optimization Problems

A Major Qualifying Project Report
Submitted to the Faculty
Of the
WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the
Degree of Bachelor of Science
For the
Industrial Engineering Program
By

_____

Uyen Ngoc Thao Nguyen

Dated:

Approved:

_____

Andrew C. Trapp, Ph.D.

_____

Renata A. Konrad, Ph.D.

Acknowledgments

I would like to thank Professor Andrew Trapp and Professor Renata Konrad, my project advisors at Worcester Polytechnic Institute, for offering me the opportunity to work on this project. I am very grateful for their prompt advice and support throughout the duration of this project.

Abstract

Conventional optimization solvers provide a single optimal solution to an optimization model, which in some cases is undesirable to the decision maker because of the large discrepancy between the optimal solution and the existing conditions, or status quo, of the real-world situation the model represents. This project focuses on developing an algorithm and computational program to generate solutions to binary integer optimization problems that can simultaneously improve the objective function value and yet control disruption from the current condition. The program uses Dinkelbach's algorithm to determine such a solution, and is implemented in Microsoft Excel utilizing Visual Basic for Applications (VBA) in conjunction with OpenSolver, an open source Excel add-in that can solve optimization problems. Detailed instructions are included to guide users through the entire process.

Contents

Table of Figures

Table of Tables

Nomenclature

BIP	Binary Integer Program
VBA	Visual Basic for Application
$\boldsymbol{x}$	Solution to the binary integer programming problem
$\boldsymbol{y}$	Current condition to the binary integer programming problem
$\boldsymbol{N(x)}$	Quality difference $N(x) = c^T x - c^T y$
$\boldsymbol{D(x)}$	Solution difference for binary integer programs $D(x) = \|x - y\|_1 = \sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i)$
$\boldsymbol{D(x)_w}$	Solution difference including variable preferences for binary integer programs $D(x) = \sum_{i:y_i=0} w_i x_i + \sum_{i:y_i=1} w_i (1 - x_i)$
$\boldsymbol{N}$	Normalization factor $N = \frac{\Delta s(x)}{\Delta q(x)}$
$\boldsymbol{\Delta s(x)}$	Interval measuring largest improving deviation of x from y $\Delta s(x) = \max D(x)$
$\boldsymbol{\Delta q(x)}$	Interval measuring largest quality improvement of x from current condition y $\Delta q(x) = \max(c^T x - c^T y)$
$\boldsymbol{\mathcal{R}(x)}$	Ratio of solution quality improvement to solution difference $\mathcal{R}(x) = \frac{N \times \Delta q(x)}{s(x)}$
$\boldsymbol{\lambda(x)}$	$\frac{N \times N(x)}{D(x)}$
$\boldsymbol{\lambda^*}$	Optimal solution for $F(\lambda)$
$\boldsymbol{F(\lambda)}$	$N \times N(x) - \lambda(x) D(x)$
$\boldsymbol{l}$	Number of desired changes (user input)
$\boldsymbol{\varepsilon}$	Numerical imprecision that naturally occurs with computer representation of small numbers; typically ε is set to 10^{-6}

Executive Summary

Background

Optimization is the process of implementing an existing situation (also referred to as current condition, or status quo), device, or system. Different optimization models can be categorized by the characteristics of the objective function, constraints, or variables they contain. The type of optimization problem that is addressed in this report, binary integer programs (BIPs), are very flexible. They can be used to model the selection or rejection of an option, a yes/no answer, and many other situations, such as planning and scheduling, distribution network, and capital budgeting. Optimization solvers typically provide a single optimal solution, which could be undesirable to the decision maker due to the large discrepancy between the provided optimal solution and the existing condition (status quo) of the modeled system. It may be beneficial to have an automated mechanism capable of generating solutions that can simultaneously improve the objective function value and control disruption from the current condition.

Project Objectives and Goals

The objectives of this project were: (1) develop an algorithmic method to generate such solutions, and (2) build an open-source computational program, integrated with a binary integer program solver, to implement this algorithmic method.

To achieve the project objective, the following goals were established:

1) Investigate solution diversity and related metrics in the context of binary integer programs (BIP);
2) Devise a methodology to find solutions similar to the status quo, but with improved objective function values;
3) Build a computational program to solve the problems above;
4) Test the proposed methodology on known problem instances.

Methodology

In Table 1 we propose a method of modifying a general binary integer programming model to address both quality improvement and disruption minimization, as well as accommodation for users to provide preferences concerning which choices should be given higher or lower disruption priority.

Table 1: Modified BIP Problem

Objective function:	$\max \frac{N \times (c^T x - c^T y)}{\sum_{i:y_i=0} w_i x_i + \sum_{i:y_i=1} w_i (1 - x_i)} = \max \frac{N \times N(x)}{D(x)_w}$
Variables	$S = \{x \in \{0,1\}^n : Ax \leq b\}$
Constraints	$c^T x - c^T y \geq \varepsilon$ $l \leq \sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i)$

In Table 1, x is the solution to the BIP problem, y is the current condition, c is the objective coefficient, N is the normalization factor, $N(x)$ is the improvement in solution quality, $D(x)_w$ is the solution difference that includes possible user-specified variable preferences, and l is the lower bound of the number of changes of x from y, respectively.

Algorithm and Program Development

As the modified objective function is not linear, one way to solve such problems is to use Dinkelbach's algorithm, which solves a sequence of simplified linearized optimization problems, iteratively converging to the optimal solution x^*.

To execute the proposed algorithm automatically, a computational program in the Microsoft Excel environment was developed by utilizing Visual Basic for Applications (VBA) to call OpenSolver, an open source Excel add-in that can solve optimization problems. A step-by-step instruction through a straightforward user-interface is presented to the user when using the program, and a trouble-shooting section was also included to assist error handling. The program requires different inputs including: current condition y, objective coefficient vector c, and variable preferences vector w. In addition, the user is able

to select a preferred number of changes of x from y within the range provided by the program. The program then works toward generating the solution to the BIP problem corresponding to the user's input.

Results

The computational program offers different tools to help users customize their binary optimization problems by adding variable preferences and setting minimum number of changes of x from y. Thus, the program can provide better choices for decision makers as they seek to find an optimal implementation of a solution to a real world problem. To test the functionality of the computational program, different examples of BIP problems were utilized, in particular the General Assignment Problem. Different scenarios to the problem were proposed based on distinct sets of variable preferences. Plots of objective values versus number of changes of solution x from current condition y were created to illustrate the trend of solution quality with respect to minimum number of changes, which is defined by the user. In addition, suggestions for future work include improving the program capability to solve the BIP minimization case, as well as making the program compatible with different solvers.

Chapter 1. Introduction

Many decisions are made with the goal of optimizing a desirable goal, for example profit, cost, or personal satisfaction. Searching for an optimal solution is a common and crucial process in a wide variety of engineering and business activities. Most engineering design involves some form of optimization in which design variables must be selected in order to achieve some sort of objective while faced with budget or functionality constraints. A simple example of a chemical engineering optimization problem is the evaluation of the optimal heat exchanger used to heat a stream (Turton, et al., 2012). Similarly, many business decisions are optimization decisions as well, where decision variables are frequently selected to maximize profit or minimize cost.

Mathematical optimization, also known as mathematical programming, is the process of finding an optimal solution from a set of feasible solutions, assuming such a solution exists. The set of feasible solutions is typically represented implicitly through specific constraints on combinations of variable values. An optimal solution is one that both satisfies these constraints and is optimal with respect to an objective function. Optimization is a powerful tool that can be used to help decision makers in circumstances relating to the allocation of scarce resources, such as investment portfolio creation, production, determining inventory levels, and many other situations.

Optimization models can be categorized by the type objective function, constraints, or variables they contain. Figure 1 provides a convenient way to classify one branch of optimization models known as mixed-integer linear programs. A mixed-integer programming problem arises when some of the variables in the model are real-valued and others are integer. Mixed-integer linear programs feature a linear objective function and linear constraints. As illustrated in Figure 1, mixed-integer linear programs with only real-valued variables are known as pure linear programs, or LPs. Pure integer linear programs (IPs) are mathematical models in which all the variables are constrained to take integer values, and are widely

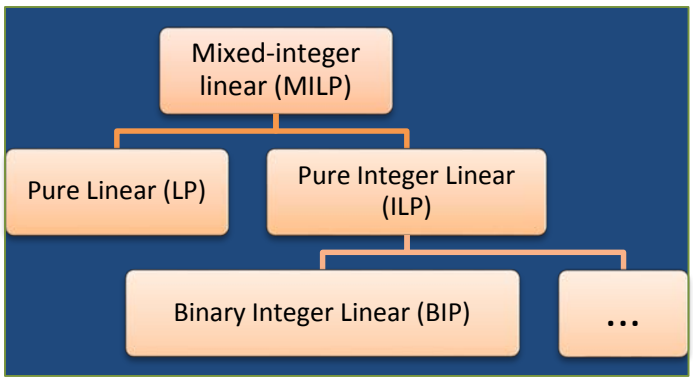


Figure 1: Hierarchy of Mixed-integer Linear Programming

used in planning and control problems such as production planning, scheduling, telecommunication networks, and cellular networks (Borndörfer & Grötschel, 2012). Integer variables that are further restricted to values of 0 or 1 are classified as binary integer variables (BIP). The binary choice enables modeling the selection or rejection of an option, a yes/no answer, and many other situations (Chinneck, 2004). The BIP optimization problem is an important class of mathematical programs and contains well-known problems such as knapsack, assignment, bin-packing, and traveling salesman (Nemhauser & Wolsey, 1998). As the focus of this project involves solutions to BIP problems, more details on BIP will be presented in Sections 2.1 and 3.1.

Traditionally, optimization solvers provide a single optimal solution; however, in certain cases, implementing the optimal solution may not be practical. For example, a Quality Control manager wishes to consolidate and redesign the lab space in a building, but is subject to a limited budget and time. If the optimal solution is too disruptive to implement, it may be better to consider an alternative that improves the objective somewhat but is easier to deploy. A second example is a project manager in a manufacturing plant wanting to increase production. However, this increase is subject to limited time, workforce, and changes in pipelines. In such an example perhaps the current production setup for workers and products have been used for several years, and shifting employees and rescheduling production according the recommendation optimal solution(s), is likely to interrupt manufacturing output and incur high costs. Hence, the project manager may wish to examine an alternative solution that produces less disruption to the current setup. In each of these examples, finding a solution that is similar to the current condition while greatly improving the objective function value may be valuable.

This project generates solutions to binary integer programs that simultaneously improve the objective function value while controlling disruption from the current condition. The objectives of this project were: (1) develop an algorithmic method to generate such solutions, and (2) build an open-source computational program, integrated with a binary integer program solver, to implement this algorithmic method. If successful, the technique could be applied to real world problems to provide better choices for decision makers.

To achieve the objectives, the following goals were established:

1. Investigate solution diversity in the context of binary integer programs (BIP);
2. Devise a methodology to find solutions not overly diverse from the status quo, but with improved objective function values;
3. Build a computational program correlated with different solvers to solve the problems above;
4. Test the proposed methodology and solver program on some known problem instances.

This project report is divided into multiple chapters that detail the developmental stages and approach of this project. This Chapter presented the problem statement, goals, and objectives of the project. Chapter 2 reviews the literature to provide a background of relevant topics. Chapter 3 discusses the methodology including techniques employed. Chapter 4 involves the algorithm and model development, and consists of four sections: (1) presenting customization and decision support options in the tool; (2) presenting Dinkelbach's algorithm to motivate solving the proposed optimization problem; (3) describing the use of VBA and OpenSolver in an Excel-based environment, together with a discussion of the overall process flow of the approach, and (4) testing the proposed approach on a known problem instance and demonstrating the results. Overall conclusions and future work are provided in Chapter 5 to review the impact of the project, while Chapter 6 presents the Industrial Engineering (IE) Reflection component.

Chapter 2. Background and Literature Review

2.1 Binary Integer Programming

Many optimization problems feature a combinatorial structure that can be modeled using binary integer variables. Examples include problems involving sequencing and selection decisions, such as distribution networks, transportation scheduling, capital budgeting, and telecommunications. These types of problems can be modeled as Binary Integer Programs (BIP). As a subset of Integer Programming [IP] (Nemhauser & Wolsey, 1998), solving BIP problems has long been an important research area, partially because of the intricacies in solution methods (Huang & Wang, 2011). In general, BIPs are *NP*-hard to solve, which means they are in a difficult class of computational programs (Karp, 2010). Until recently, the solution to many BIPs were beyond the capability of state-of-the-art solvers, and many still remain so. We next discuss three common approaches to solve binary integer programs.

2.1.1 Exact Approach

Exact approaches include exhaustive search, which enumerates all potential solutions from a given set and selects the best one (Boswell, 2012). This enumeration can be completed explicitly or implicitly. The branch-and-bound algorithm represents the latter, systematically enumerating all possible solutions, and discarding potentially large subsets of unqualified candidates altogether based on the upper and lower estimated bounds of the optimized value (Nemhauser & Wolsey, 1998). Dynamic programming is another form of exhaustive search that precludes unnecessary re-computation by storing the solutions of sub-problems. This technique utilizes the solution process as a recursion (Bellman, 2003).

2.1.2 Approximation Algorithms

Many important computational problems are challenging to solve to optimality. These include those that are *NP*-hard, such as the Traveling Salesman Problem (TSP), Vertex Cover, Max Clique, and many more (Chakrabarti, 2005; Gomes & Williams, 2005). Instead of using exhaustive search, which may be

prohibitive due to the consideration of all possible solutions, approximation algorithms are a useful approach to help find estimated solutions to such difficult problems.

An approximation algorithm generally has two properties:

1. Provides a method for obtaining a feasible solution (if one exists) in polynomial time;
2. Assures the objective quality of the solution, by providing a bound on the maximum "distance" between the approximate solution and an optimal solution.

By considering all possible instances of an optimization problem, approximation algorithms solve an optimization problem approximately, within a factor bounded by a constant or by a slowly growing function of the input size (Gomes & Williams, 2005). Three common techniques for this algorithm are witnesses, and coarsening and relaxation, (Klein & Young, 1999). The witness method encrypts a short, simple-to-verify proof that the optimal value is approximately a certain value, providing a dual role to possible solutions to a problem. In the case of a maximization problem, relaxation is an alternative way to obtain an upper bound on the maximum value. Coarsening represents a relatively broad category of algorithms that replace the original problem with a less complex one, while ensuring a rough correspondence between the feasible solutions of the two (Klein & Young, 1999).

Relaxation is an alternative way to obtain bounds for the objective value. For instance, in the case of a binary integer program, linear relaxation methods replace the original restriction that each variable must be 0 or 1 by a weaker constraint in which each variable belongs to the interval [0, 1]. This technique can convert an *NP*-hard binary integer programming problem into a linear program, which is solvable in (pseudo-)polynomial time. Solutions to the relaxed linear program can be used to understand more about the solutions to the original binary integer program (Matoušek & Gärtner, 2007) and bounds the integer program's optimal objective function value (Hochbaum, 2008). This relaxation may be solved using standard linear programming techniques such as the simplex algorithm of Dantzig (Dantzig, 1951), interior point methods, Criss-cross algorithm (Fukuda & Terlaky, 1997), and the conic sampling algorithm of

(Serang, 2012), among others. Linear programming relaxation is a standard technique for designing approximation algorithms for hard optimization problems.

Still other approximation algorithm methods include the small-additive-error algorithm (Roditty & Shapira, 2011), randomized rounding (Thai, 2013), polynomial approximation schemes, the constant-time algorithm (Nguyen & Onak, 2008), among others.

2.1.3 Heuristic Algorithms

Even though exact approaches are quite efficient, there are several drawbacks with each method. For example, branch-and-bound and dynamic programming methods are often time-consuming (Kokash, 2005). Heuristic algorithms, on the other hand, can often find good solutions quickly, at the expense of a guarantee on optimality. Generally speaking, heuristics can produce adequate solutions but do not offer a guarantee on the quality of their solutions. This is in contrast to approximation algorithms, which may produce a solution that is assured to be within some factor of optimal. One type of heuristic is the greedy algorithm, which is based on the principle of taking the best local values with the hopes of finding a global optimum to the objective function (Cormen , et al., 2009). Another popular heuristic is local search, which focuses on a subset of the solution domain. Local searches can be categorized in different ways, such as the hill-climbing technique, which considers neighboring solutions and replaces the current condition if the neighbor has a superior objective value (Stutzle, 1998). Similar to the hill-climbing approach, the simulated annealing algorithm occasionally accepts solutions that are worse than the current condition as its acceptance probability get decreased over time (Aydin & Fogarty, 2004). However, it provides more variability at the beginning of search, and the probability of picking move is related to how good it is. Likewsie, the Tabu Search avoids local optima by using memory structures to forbid the recurrence of moves that have been executed recently (Battini, 1996). In a similar vein, swarm intelligence methods are recognized as an artificial intelligence technique which can be impressively resistant to the problem of local optima (Eberhart, et al., 2001).

2.1.4 Binary Integer Problem Solvers

Multiple exact solvers exist that can solve general binary integer programs, including CPLEX Optimizer (IBM, 2014), Gurobi (Gurobi, 2014), OpenSolver/ CBC (Lougee-Heimer, 2010), the Excel Solver (Fylstra, et al., 1998), MATLAB (Mathworks, 2014), and many more. The solvers most relevant to this study are:

OpenSolver:

OpenSolver is an open-source linear and integer optimization add-in for Microsoft Excel. Separate cells are used to represent decision variables, while formulas are placed in user-designated cells to represent the objective function, and constraints. OpenSolver uses high-powered COIN-OR/CBC algorithms to solve linear and integer programming problems. It provides some enhanced capabilities over the standard Excel Solver, including a visualizer that highlights the model's decision variables, objective and constraints directly on one's spreadsheet (Lougee-Heimer, 2010). In particular, there are no restrictions to the number of variables or constraints in a given model.

Excel Solver:

As mentioned above, Microsoft Excel has an add-In optimization tool, Solver, which solves linear, integer, and nonlinear programming problems. Like OpenSolver, separate cells are used to represent decision variables, while formulas are placed in user-designated cells to represent the objective function and constraints. Once the model is entered in a spreadsheet and declared via the Solver window, Solver can be called to find the solution (Fylstra, et al., 1998).

Furthermore, other common solvers include the following:

CPLEX Optimizer:

CPLEX, free for academic use, is recognized as a popular solver with an API for several programming languages. This solver utilizes robust algorithms to solve problems with up to millions of constraints and variables, and parallelizes robustly. CPLEX provides flexible, high-performance mathematical programming solvers for optimization problem classes that include linear programming, mixed-integer

programming, quadratic programming, and quadratically constrained programming problems, among others (IBM, 2014).

Gurobi:

Gurobi is a more recent addition to the optimization solver playing field, founded in 2008. It is also free for academic use, supports parallel processing, and can solve the following types of problems: large-scale linear programs (LP), quadratic programs (QP), quadratically constrained programs (QCP), mixed-integer linear programs (MILP), mixed-integer quadratic programs (MIQP), and mixed-integer quadratically constrained programs (MIQCP) (Gurobi, 2014).

MATLAB Optimization Toolbox:

Optimization Toolbox offers standard algorithms to solve constrained and unconstrained continuous and discrete optimization problems. Some of the mathematical optimization problems it can solve are linear programs, quadratic programs, and other nonlinear programs. It also is capable of parallel computing through a supplemental toolbox (Mathworks, 2014).

2.1.5 Utilizing OpenSolver

As the focus of this project is solving BIPs in Excel, a solver that is compatible with Excel should be chosen. In light of OpenSolver's ability to solve large optimization models using COIN-OR/CBC software, together with the lack of any restriction on the number of model variables and constraints, we elected to use OpenSolver for this project. In addition OpenSolver supports Visual Basic for Applications (VBA) functionalities, which enable customization for a computational program.

2.2 Solving Binary Integer Programs with More than One Objective

Exact solvers typically generate a single optimal solution that optimizes the objective function value while satisfying the given constraints. However, sometimes such an optimal solution is undesirable to the decision maker as it is too diverse from the status quo. For instance, a hospital director would like to

change the work schedule for nurses and doctors to best meet the needs of the patients, while satisfying staff preferences and minimizing operating costs. However, an optimal solution to the scheduling problem may require such an enormous upheaval that could render the optimal solution unlikely to be implemented. In such a situation, the hospital director might instead like to know whether there are any alternatives that are sufficiently similar to the status quo, and yet yield a solid improvement in terms of meeting patient, staffing preferences and costs. Thus, in certain circumstances, solutions that do not overly disrupt t the status quo but improve the objective function value, could be beneficial to decision makers.

Many studies have focused on the importance of multiple diversity and high-quality solutions including mixed-integer programming (Murthy, et al., 1999; Glover , et al., 2000; Danna & Woodruff, 2009), and binary integer programming (Trapp & Konrad, 2013). Some studies discuss the benefit of high-quality solutions generated in a short amount of time (Boland , et al., 2013), while other studies include a focus on solution diversity (Murthy, et al., 1999; Danna & Woodruff, 2009) . Although a number of efficient approaches are available to provide high-quality and/or diverse solutions for optimization problems, very few studies focus on controlling deviation from a current condition while improving the objective function value. As highlighted above, this type of tool may prove very valuable to practical problems faced in industry.

2.3 Pareto Optimization

At times the decision maker may be interested in the trade-off between two objectives such as improving the BIP objective while minimizing the disruption from current condition. Optimization problems that consider more than one objective, such as minimizing solution disruption and maximizing objective function quality, are often referred to as a multi-objective optimization (also known as Pareto optimization). Solutions to multi-objective problems rarely meet all objectives simultaneously. A possible compromise is through the concept of Pareto optimality. Pareto optimal solutions cannot be improved in any of the objectives without reducing the value of at least one of the other objectives. Multi-objective problems typically have numerous Pareto optimal solutions; various approaches have been devised to translate the multiple objectives into a single objective to locate such solutions. For instance, Figure 2 presents the set of Pareto optimal solutions denoted in a bold line, where the largest value in one objective occurs in either point $\left(f_2(\widehat{x}), f_1(\widehat{x})\right)$ or $(f_2(\tilde{x}, f_1(\tilde{x}))$. Some methods to solve such problems include scalarizing multi-objective optimization (Hwang & Masud, 1979), genetic algorithms (Konak, et al., 2006), and other metaheuristic approaches (Fonseca & Fleming, 1993). In our project, a new binary integer programming model is introduced to provide solutions that can simultaneously improve the current condition's objective value and control the number of changes from the current condition.

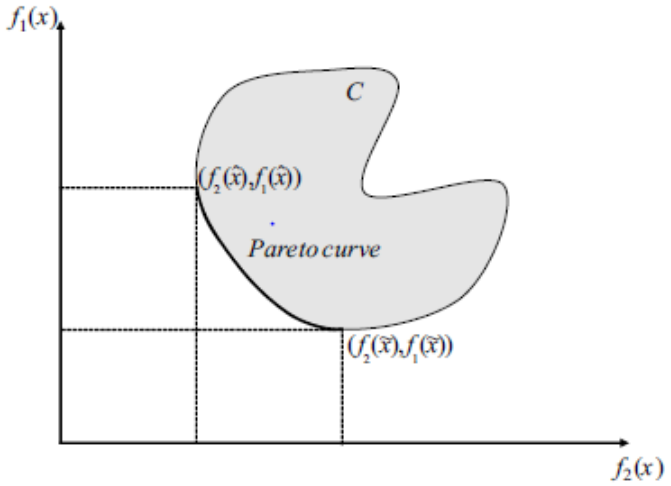


Figure 2: Example of Pareto curve (Caramia & Dell'Olmo, 2008)

Chapter 3. Methodology

In this section, a detailed mathematical description is introduced to generate solutions to a specific binary integer program that is formulated with respect to another reference binary integer program. This new binary integer program has the property of controlling deviation from the current condition to the reference BIP while at the same time balancing the competing goal of improving its objective function.

3.1 Problem Description

A binary integer linear program is comprised of a vector $c \in \mathbb{R}^n$, a right hand side vector $b \in \mathbb{R}^m$, and a matrix $A \in \mathbb{R}^{m \times n}$. The general BIP is stated as in (1)-(2); where without loss of generality we assume the maximization case:

$$\max \{c^T x | x \in S\} \tag{1}$$

$$\text{where } S = \{x \in \{0,1\}^n : Ax \leq b\} \tag{2}$$

The feasibility set S is bounded and for convenience of describing the methodology, it is assumed to be nonempty, i.e., $S \neq \emptyset$. Let y be a current condition to the problem that represents status quo, and $c^T y$ be its objective function value. To make the discussion on finding solutions closest to x relevant, we assume that at least one alternate feasible solution in addition to y in S exists, that is, $S \backslash \{y\} \neq \emptyset$.

3.2 Modified Binary Integer Programming Model

To address different objective functions including solution quality and solution similarity at the same time, a general BIP is modified as shown in the sections below.

3.2.1 Addressing Solution Similarity

Measuring the difference between two binary vectors x and y can be accomplished through a variety of methods, such as L_1 (taxicab) norm or the L_2 (Euclidean) norm. For example, the L_1 norm measures the solution similarity between x and y in (3) using absolute values:

$$solution\ difference = \ D(x) = \|x - y\|_1 = \sum_i^n |x_i - y_i|. \quad (3)$$

It turns out that, for binary vectors, the L_1 norm can be equivalently represented as (Balas & Jeroslow, 1972):

$$D(x) = \|x - y\|_1 = \sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i) \quad (4)$$

3.2.2 Addressing Solution Quality

For any $x \in S$, the objective function $c^T x$ expresses its quality. For any $x \in S$, the difference in solution quality with respect to x can be expressed as:

$$quality\_improvement = \ N(x) = c^T x - c^T y \quad (5)$$

If the difference in (5) is positive, then the objective function value is improved and therefore is of higher quality.

3.2.3 Addressing Both Solution Similarity and Quality

One way to handle the competing objectives of maintaining similarity between x and y while improving the quality of x is via a ratio measure:

$$\mathcal{R}(x) = \frac{relative\ improvement\ in\ solution\ quality}{relative\ deviation\ from\ initial\ solution\ y} \quad (6)$$

With

$$N = normalization\ factor = \frac{\Delta s(x)}{\Delta q(x)}$$

$$= \frac{Interval\ measuring\ largest\ improving\ deviation\ of\ x\ from\ y}{Interval\ measuring\ largest\ quality\ improvement\ of\ x\ from\ y} \quad (7)$$

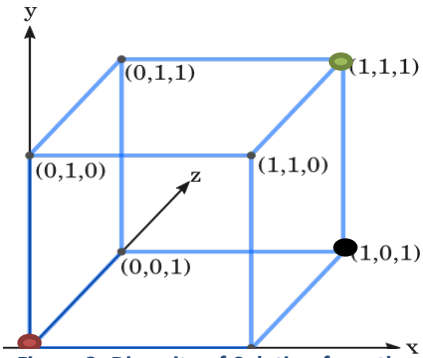


Figure 3: Diversity of Solution from the Current Condition y

To calculate $\Delta q(x)$, we only consider improvement in the objective function. For example, if there is a large difference in $c^T x$ and $c^T y$, but the value of the objective is lower, $c^T x < c^T y$, x will not be considered as a solution in the calculation of $\Delta q(x)$. This is further illustrated in Figure 3, which contains a three-dimensional cube with current condition $y = (1,0,1)$. Assume the corresponding objective function is:

$$\max(x_1 + x_2 + x_3)$$

Feasible solutions to the problem are found at $(1,1,1)$, which strictly improves the objective function, while the solution $(0,0,0)$ strictly decreases the objective function. To ensure the objective function value improves, $(0,0,0)$ will not be considered even though it has greater diversity from y than $(1,1,1)$.

Thus, quality improvement interval, $\Delta q(x)$, can be presented as

$$\Delta q(x) = \max quality\_improvement - \min quality\_improvement, \quad (8)$$

Or equivalently,

$$\Delta q(x) = \max N(x) - \min N(x). \quad (9)$$

Because

$$\min N(x) = 0 \quad (10)$$

represents no improvement in quality, hence,

$$\Delta q(x) = max\, N(x) = max(c^T x - c^T y) \quad (11)$$

In addition, the largest improving deviation of x from y, $\Delta s(x)$ can be calculated as

$$\Delta s(x) = \max solution\_difference - \min\ solution\_difference \quad (12)$$

Or equivalently,

$$\Delta s(x) = \max D(x) - \min D(x). \quad (13)$$

By definition,

$$\min D(x) = 0 \ (e.g., when \ \ x = y), \tag{14}$$

Thus this gives

$$\Delta s(x) = \max D(x). \tag{15}$$

From equations (3),(4), and (15), the solution difference can be expressed as:

$$\Delta s(x) = \max D(x) = \max \left[\sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i) \right] \tag{16}$$

Putting together (11) and (16), we now have:

$$N = \frac{\Delta s(x)}{\Delta q(x)} = \frac{\max \sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i)}{\max(c^T x - c^T y)}. \tag{17}$$

From (4), (5), (6), and (17), we can now formulate the objective function as:

$$\max \mathcal{R}(x) = \max \frac{N \times (c^T x - c^T y)}{\sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i)} = \max \frac{N \times N(x)}{D(x)} \tag{18}$$

Again, the intent was to find solutions to a BIP problem instance that control the deviation from a status quo solution y while improving the objective function value. Accordingly, constraints were added to the model to ensure the improvement in quality (19) and at least one change from y was made (20).

$$c^T x \geq c^T y \tag{19}$$

$$\sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i) \geq 1 \tag{20}$$

Table 2 summarizes the modifications to a given binary integer program.

Table 2: Modified Binary Integer Problem for Multiple Objective Functions

Objective function:	$\max \frac{N \times (c^T x - c^T y)}{\sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i)} = \max \frac{N \times N(x)}{D(x)}$
Variables	$S = \{x \in \{0,1\}^n : Ax \leq b\}$
Constraints	$c^T x - c^T y \geq 0$ $\sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i) \geq 1$

Chapter 4. Model Development and Algorithm

Given a binary integer program and a feasible solution y, and incorporating the modifications detailed in Table 2, we now discuss about the customization to the modified BIP problem based on user preferences (Table 2), as well as a method to solve this optimization problem.

4.1 Customization Option and Decision Support

To extend the flexibility of the program, several modifications to the model were proposed. Initially we considered only that the number of changes from status quo, y, to the BIP solution, x, must be greater than or equal to 1. However, as the model tended to minimize number of changes from y to x to trade off with an increase in objective function value, sometimes the computational program would provide solutions that were not much different from the status quo, which may not be desirable to the decision maker. Thus, we propose an option allowing the user to choose a preferred lower bound number of changes from y to x, which modify constraint (20) to

$$\sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i) \geq l \tag{21}$$

A second option was to weigh the importance of particular decision variables. In reality, one decision variable might not necessarily carry the same level of importance as another to the decision maker. For example, in a healthcare scheduling problem it is often the case that changing a physician's assignment is more difficult to implement than a nurse's assignment. In such cases changing x_i (a physician's assignment) may have a greater impact than changing x_j (a nurse's assignment). Therefore, the weighting of the variable vector x should be considered in evaluating the solution deviation to improve the practicality of the model. This could be done by adding weights w_i, for all variables x_i to account for the difficulty of changing x_i from y_i. Thus, this modification defined the new objective function as:

$$\max \frac{N \times (c^T x - c^T y)}{\sum_{i:y_i=0} w_i x_i + \sum_{i:y_i=1} w_i (1 - x_i)} = \max \frac{N \times N(x)}{D(x)_w} \tag{23}$$

where w is the vector of variable preferences and $D(x)_w$ is a modified solution difference with variable preferences. For instance, if all variables are considered to have the same weighting (or same preferences), the w vector can be presented to be all 1's.

Furthermore, to ensure the functionality of the proposed model, feasibility of the current condition y was examined before solving the model by checking if it satisfies the model constraints, that is,

$$Ay \leq b. \tag{24}$$

Also, ε was also added to the quality improvement constraint to help with the numerical imprecision that naturally occurs with computer representation of these small numbers, where $\varepsilon = 10^{-6}$. These proposed modifications to the model and computational program improved the usefulness of the tool. The customized model is summarized as in Table 3.

Table 3: Modified BIP Problem

Objective function:	$\max \frac{N \times (c^T x - c^T y)}{\sum_{i:y_i=0} w x_i + \sum_{i:y_i=1} w (1 - x_i)} = \max \frac{N \times N(x)}{D(x)_w}$
Variables	$S = \{x \in \{0,1\}^n : Ax \leq b\}$
Constraints	$c^T x - c^T y \geq \varepsilon$ $l \leq \sum_{i:y_i=0} x_i + \sum_{i:y_i=1} (1 - x_i)$

4.2 Dinkelbach's Algorithm

The proposed objective function for BIP, as in (23), is a fractional, or hyperbolic, binary integer program. To find a single optimal solution for this objective function, Dinkelbach's algorithm can be used to solve the fractional binary program in (23) (Trapp & Konrad, 2013). Details of Dinkelbach's algorithm can be found in (Dinkelbach, 1967; Schaible, 1976).

For the BIP model proposed for this project in Table 3, consider the objective function as:

$$\max\left\{\lambda(x) = \frac{N \times N(x)}{D(x)_w} | x \in S\right\}, \tag{25}$$

which maximizes function $\lambda(x)$ by balancing the competing goals of maximization in quality improvement and minimization in solution deviation of x from y. N is the normalization factor from (17), $N(x)$ is the difference in solution quality as in (5) , and $D(x)$ is the solution difference in (4).

From the related theory (Dinkelbach, 1967), $\lambda(x)$ is maximized if and only if

$$\max\{N \times N(x) - \lambda D(x)_w | x \in S\} = 0 \tag{26}$$

For an optimal value λ^*.

Dinkelbach's algorithm solves a sequence of linearized problems that are related to the original nonlinear fractional programming problem to find the optimal solution x^* and corresponding optimal value λ^*. Building on this foundation, Dinkelbach's algorithm for the BIP problem is as follows:

Input: Feasibility set of current condition $y \in S$, variable preferences w, normalization factor N, solution difference $D(x)$, and solution quality improvement $N(x)$.

Output: Optimal solution x^*.

Step 1: Set $k = 0$, and choose initial starting value for λ^0 (e.g., $\lambda^0 = 0$).

Step 2: Solve optimization problem (26), which is $\max\{N \times N(x) - \lambda^k D(x)_w | x \in S\}$, and denote optimal solution as x^k.

Step 3: If $\left|N \times N\left(x^k\right) - \lambda^k D\left(x^k\right)\right| \leq \varepsilon$, **set** $x^* = x^k$, and **STOP**. Otherwise, compute $\lambda^{k+1} = \frac{N \times N(x^k)}{D(x^k)_w}$, let $k = k + 1$, and go back to step 2.

The algorithm exits when $\left|N \times N\left(x^k\right) - \lambda_{k+n} D\left(x^k\right)\right| \leq \varepsilon$ with x^* as the optimal solution that maximizes (25).

4.3 VBA and Solvers in the Excel Environment

Dinkelbach's algorithm proposed in Section 4.2 typically requires multiple iterations to find an optimal solution. Thus, to solve the proposed BIP problem summarized in Table 3, we developed a computational

program in Microsoft Excel to automate the algorithm. Our approach used Visual Basic for Applications (VBA) (Microsoft, 2013) to call OpenSolver (Lougee-Heimer, 2010) to optimize all sub-problems occurring in the algorithm outlined in Section 4.1 and 4.2.

The user interface of the program is shown in Figure 4. It includes detailed instructions on each step that the user can easily follow to conduct our approach on their own binary integer program.

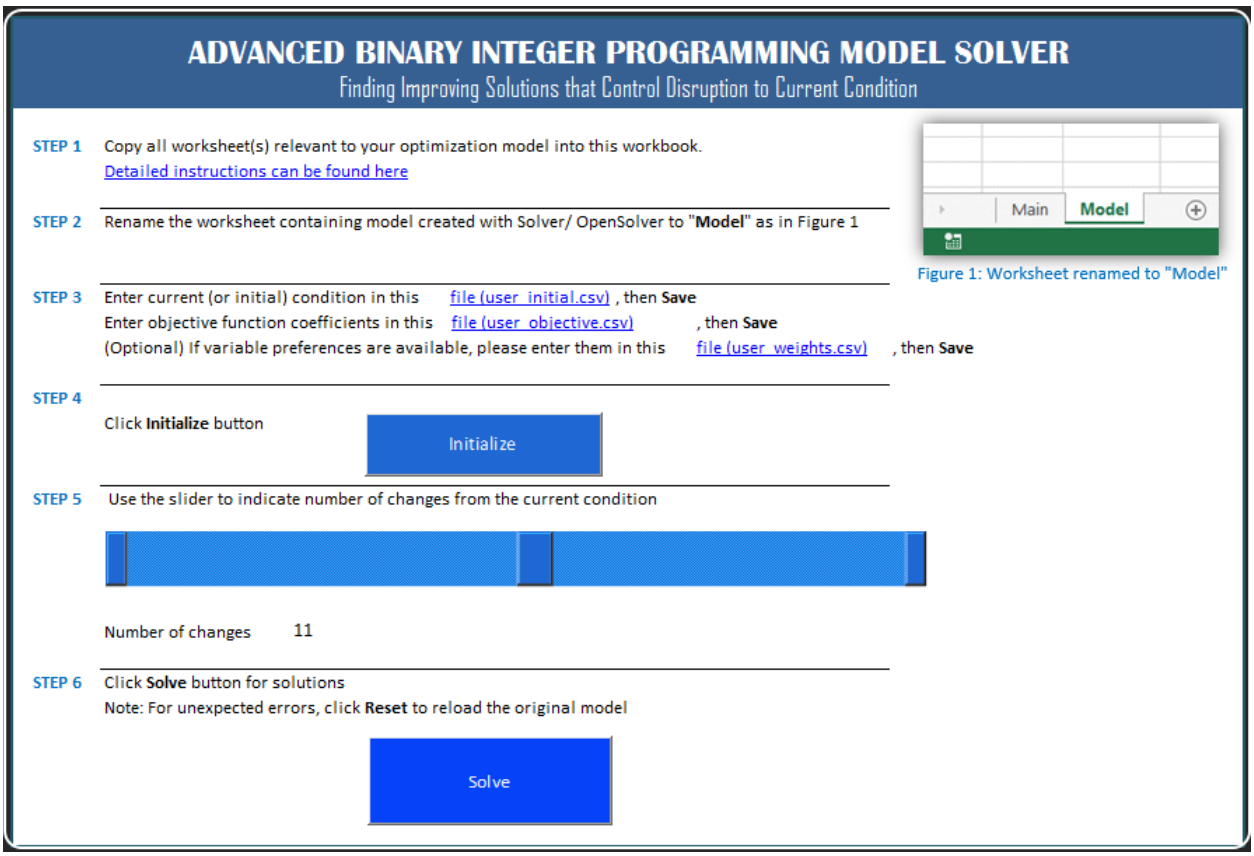
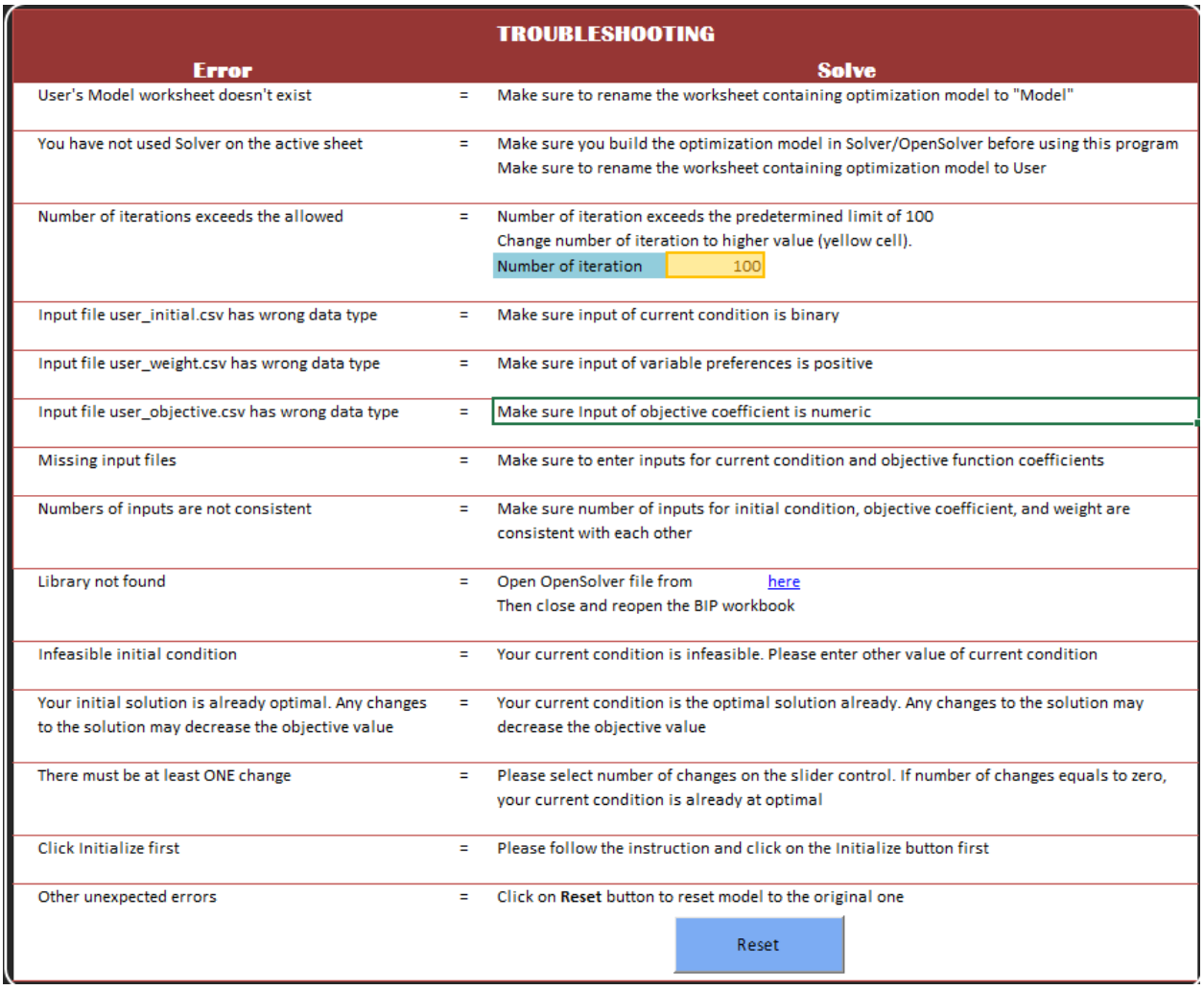


Figure 4: User Interface of Program in Excel

There is also a troubleshooting section provided as shown in Figure 5, which includes all the interpretations of all common error messages and suggested resolutions. Finally, there is a hard Reset button to help users retrieve their original model whenever desired.



TROUBLESHOOTING		
Error		**Solve**
User's Model worksheet doesn't exist	=	Make sure to rename the worksheet containing optimization model to "Model"
You have not used Solver on the active sheet	=	Make sure you build the optimization model in Solver/OpenSolver before using this program Make sure to rename the worksheet containing optimization model to User
Number of iterations exceeds the allowed	=	Number of iteration exceeds the predetermined limit of 100 Change number of iteration to higher value (yellow cell). Number of iteration 100
Input file user_initial.csv has wrong data type	=	Make sure input of current condition is binary
Input file user_weight.csv has wrong data type	=	Make sure input of variable preferences is positive
Input file user_objective.csv has wrong data type	=	Make sure Input of objective coefficient is numeric
Missing input files	=	Make sure to enter inputs for current condition and objective function coefficients
Numbers of inputs are not consistent	=	Make sure number of inputs for initial condition, objective coefficient, and weight are consistent with each other
Library not found	=	Open OpenSolver file from here Then close and reopen the BIP workbook
Infeasible initial condition	=	Your current condition is infeasible. Please enter other value of current condition
Your initial solution is already optimal. Any changes to the solution may decrease the objective value	=	Your current condition is the optimal solution already. Any changes to the solution may decrease the objective value
There must be at least ONE change	=	Please select number of changes on the slider control. If number of changes equals to zero, your current condition is already at optimal
Click Initialize first	=	Please follow the instruction and click on the Initialize button first
Other unexpected errors	=	Click on **Reset** button to reset model to the original one Reset

Figure 5: Troubleshooting Section

Overall, the user will follow the sequence of steps as shown in Figure 6.

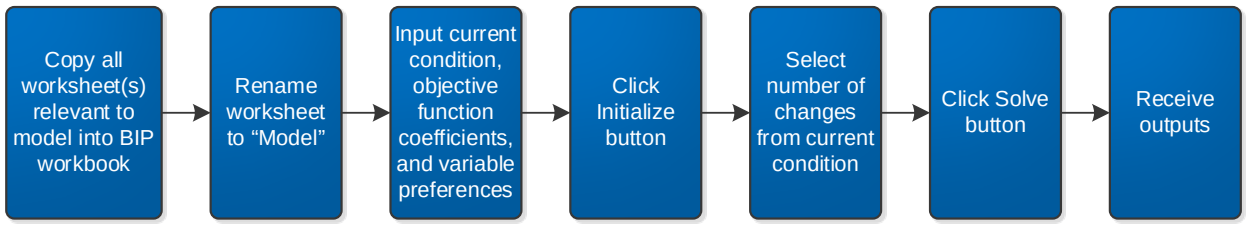


Figure 6: Overall User Experience Flowchart

First, the user is required to copy all worksheet(s) relevant to the user's model into the BIP workbook and instructions are provided. The user will then need to rename the worksheet containing the model in Solver or OpenSolver into "Model". After that, the user will be instructed to input the status quo y, the variable

preferences w, and the objective function coefficients c into the program. After the user clicks Initialize button, user will be able to choose the lower bound number of changes of solution x from current condition y. By clicking the Solve button, the program will run the algorithm, calling OpenSolver to optimize the model, and retrieving the results from the BIP program. The functionality of the Initialize and Solve buttons, as well as troubleshooting error messages, are further discussed in Sections 4.3.1 - 4.3.3.

4.3.1 Initialize Button

The process map of the Initialize button is illustrated in Figure 7. After the user clicks on the Initialize button, the program will first check for the existence of the "Model" sheet through the first two steps as illustrated in Figure 6. If the "Model" sheet is available, the program will then determine whether the worksheet contains the BIP model in Solver or OpenSolver. If the model is available, a copy will be stored to be loaded back into Solver through the Reset button. Then, the program will read three types of user input from the input files and validate their compatibility with each other and with the model variables. If all conditions required for a BIP model and inputs are satisfied, the maximum and minimum differences between the solution and the status quo to increase objective value from its current condition will be calculated internally, and subsequently displayed on the slider control for the user to select.

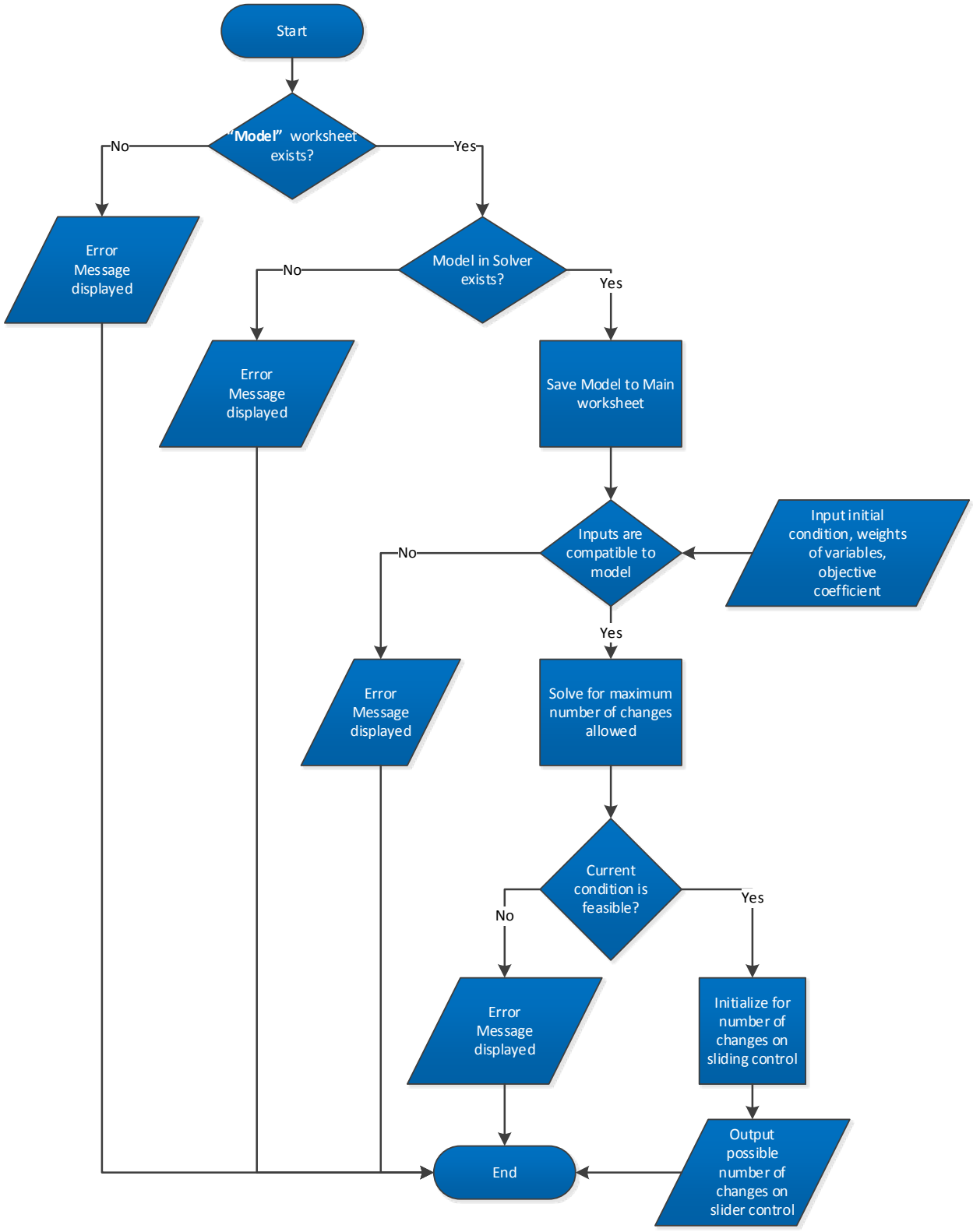


Figure 7: Process Map of the Initialize Button in VBA

4.3.2 Solve Button

Once the model is initialized, the user will be able to select the desired minimum number of changes of the solution x from the status quo y through the slider control bar. By clicking the Solve button, the BIP problem in Table 3 will be solved by applying our implementation of Dinkelbach's algorithm which calls OpenSolver, as shown in Figure 8. After termination, the solutions to the problem including the current (original) objective function value, objective function value improvement, number of changes, and number of iterations in Dinkelbach's algorithm will be outputted in a pop up window. The VBA code is included in Appendix A.

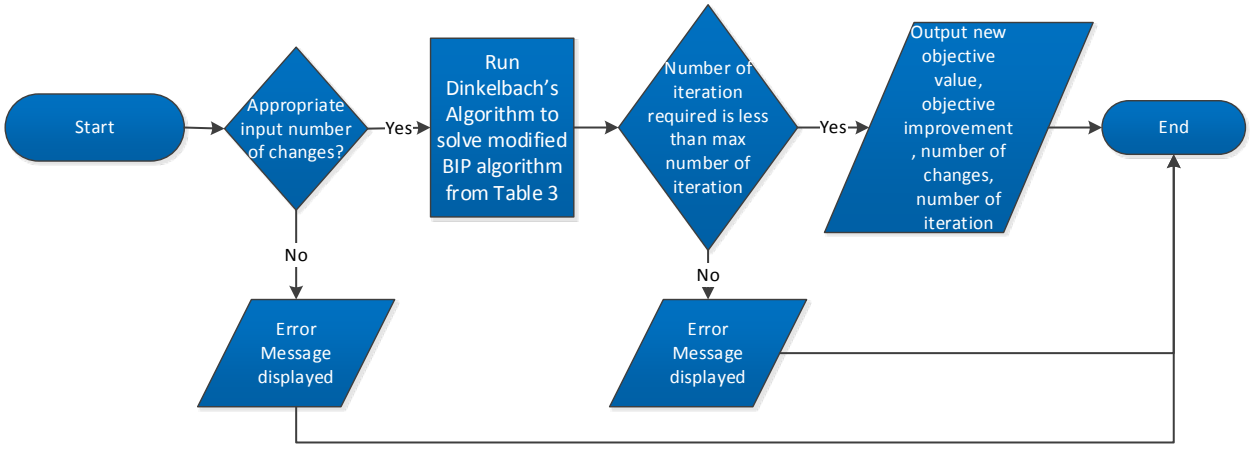


Figure 8: Process Map of Solve Button in VBA

4.3.3 Error Messages and Troubleshooting

As mentioned in Sections 4.3.1 and 4.3.2, to ensure the functionality and the quality of the computational program, different tests on the inputs are conducted. Table 4 lists possible user-generated errors and troubleshooting messages which will be displayed to the user.

Table 4: Input Errors and Associated Troubleshooting Tips

Events	Errors	Troubleshooting
Initialize Button/ Solve Button	Model worksheet doesn't exist	Make sure to rename the worksheet containing the optimization model to “Model”
Initialize Button/ Solve Button	Solver has not been used on the active sheet	Make sure to build the optimization model in Solver/OpenSolver before using this program Make sure to rename the worksheet containing optimization model to “Model”
Initialize Button	Numbers of inputs do not have consistent dimension	Ensure that the number of inputs for initial condition, objective coefficient, and weight have consistent dimension with one another
Initialize Button	Initial solution is already optimal. Any changes to the solution may decrease the objective value	Current condition is already optimal . Any changes to the solution may decrease the objective value
Initialize Button	Infeasible initial condition	Current condition is infeasible. Please enter another value for the current condition
Initialize Button	Input file user_initial.csv has an incorrect data type	Make sure the input for the current condition is binary
Initialize Button	Input file user_weight.csv has an incorrect data type	Make sure the input for variable preferences is positive
Initialize Button	Input file user_objective.csv has an incorrect data type	Make sure the input of objective coefficient has a numeric input
Solve Button	There must be at least ONE change	Please select number of changes on the slider control. If the number of changes equals to zero, the current condition is already optimal
Solve Button	Click Initialize first	Please click on the *Initialize* button first (as indicated in the Instructions)
Solve Button	Number of iterations exceeds the allowed amount	The number of iterations exceeds the predetermined limit of 100. Change the number of iteration to higher value in yellow cell, where the default number of iteration was specified to be 100.

4.4 Examples and Discussion

To demonstrate the functionality and features of the program, a sample binary integer program from the literature, known as a Generalized Assignment Problem [GAP], is presented in this section (Cattrysse, et al., 2003). The mathematical formula, results and discussion of the program performance are provided below.

4.4.1 Mathematic Formulation

The objective of a general assignment problem objective is to assign m tasks to n machines typically to maximize profit such that each task is scheduled to only one machine and machine can generally handle more than one tasks. The conventional formulation for the general assignment problem is:

$$\text{Max } z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{27}$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ with } i = 1,2,\dots,m \ (assignments) \tag{28}$$

$$\sum_{i=1}^{m} z_{ij}x_{ij} \leq d_j \text{ where } j = 1,2,\dots,n \ (machine\ capacity) \tag{29}$$

$$x_{ij} \in \{0,1\}^{m\times n} \tag{30}$$

Where c_{ij} is the profit obtained if task i is assigned to machine j, d_j is the capacity of machine j, and z_{ij} is the amount of resource required of machine j required to complete task i. Note that z represents the objective function to be maximized.

4.4.2 Input

As discussed in Section 4.3, the solution process requires the user to input his or her model, the current condition y_{ij}, the objective coefficients c_{ij}, and the variable preferences w_{ij}. The model used in this example is shown in Figure 9 below and described in the following text.

User's Model: Screen shots of the input from the user's GAP model are shown in Figure 9. In total, the problem has 75 decision variables and is subject to 20 constraints. The top matrix in Figure 9 corresponds to objective function coefficients, the second matrix corresponds to the amount of resource of each machine required to complete each different task, and the third matrix presents the decision variables, which are binary and restricted by constraints (28) and (29). The objective function cell seeks to maximize profit.

Generalized Assignment Problem 1 (GAP1)

Profit Obtained from Allocating Job j to Agent i

Machine/ Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	17	21	22	18	24	15	20	18	19	18	16	22	24	24	16
2	23	16	21	16	17	16	19	25	18	21	17	15	25	17	24
3	16	20	16	25	24	16	17	19	19	18	20	16	17	21	24
4	19	19	22	22	20	16	19	17	21	19	25	23	25	25	25
5	18	19	15	15	21	25	16	16	23	15	22	17	19	22	24

Resources Consumed Allocating Job j to Agent i

Machine/ Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	8	15	14	23	8	16	8	25	9	17	25	15	10	8	24
2	15	7	23	22	11	11	12	10	17	16	7	16	10	18	22
3	21	20	6	22	24	10	24	9	21	14	11	14	11	19	16
4	20	11	8	14	9	5	6	19	19	7	6	6	13	9	18
5	8	13	13	13	10	20	25	16	16	17	10	10	5	12	23

Maximize Profit

334

Machine/ Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			Total Resources Available
1	0	0	0	0	1	0	1	0	1	0	0	0	0	1	0	33	<=	36
2	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	27	<=	34
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	38	<=	38
4	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	27	<=	27
5	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	28	<=	33
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			
	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=			
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			

Figure 9: Example of Sample General Assignment Problem Model Inputted into the Computational Program

Current Condition

The current condition to the problem is inputted and displayed in the csv file as shown in Figure 10. The user can enter the current condition as a matrix, a single row, or a single column as long as the association of the variable order in the solution as well as the current condition are identical. For instance, with respect to the variables range, the current condition y_{ij} must be entered from left to right, and then from top to bottom. Thus, the user can choose to have the current condition entered with the same layout as the variable range.

1	0	0	0	0	0	1	0	1	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	0	0	1	0	0

Figure 10: Current Condition to the Sample GAP Problem

Objective Function Coefficients

Similar to the input method for the current condition, the objective function coefficients can be entered as a matrix, a single row, or a single column. Additionally, if the objective coefficient is already displayed in the "Model" sheet, the user can simply copy the data right into the csv input file. In this case, the user chose to copy the objective coefficient from the original model to the input file, and shown in Figure 11.

17	21	22	18	24	15	20	18	19	18	16	22	24	24	16
23	16	21	16	17	16	19	25	18	21	17	15	25	17	24
16	20	16	25	24	16	17	19	19	18	20	16	17	21	24
19	19	22	22	20	16	19	17	21	19	25	23	25	25	25
18	19	15	15	21	25	16	16	23	15	22	17	19	22	24

Figure 11: Objective Function Coefficients for the Sample GAP Problem

Variable Preferences

The user has an option to input variable preferences to prioritize which decision variables should change. By default, all preferences are set to a value of one. Two variable preference scenarios w illustrate the impact of this option on the solution.

Scenario 1: The assignment of tasks to machines had equal preferences. Each decision variable x_{ij} had a default preference of one, i.e., $w_{ij} = 1$.

Scenario 2: It was assumed that the assignment of certain tasks i to particular machines j were particularly disruptive to the status quo and thus undesirable. In such a case, the variable preferences for these particular assignments had a user-inputted value greater than one as illustrated in Figure 12 (for example x_{21} has a preference $w_{ij} = 3$). The user can select the scale of preferences; however in all cases a higher value relative to others indicates that changing the value of the current decision variable from y_{ij} is less desirable. In the example a value of three corresponds to the most disruptive (least desirable) change, while a value of one corresponds to a change that is less undesired.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	1	2	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	2	1	1	1	1	2	1	1
1	1	2	1	1	1	1	1	1	1	3	1	1	2	1
1	1	1	1	1	1	1	3	1	1	1	1	1	1	1

Figure 12: Variable Preferences for Sample GAP Problem

4.4.3 Solution to the Binary Integer Programming Problem

Once the user enters all required inputs into the model, the Initialize button is then clicked. At this point the absence of error messages shown indicates that the BIP model was correctly saved into the Main worksheet. The error-checking process also confirms that the current condition is feasible and satisfies all model constraints. Next, OpenSolver determines the maximum number of changes and the optimal objective function value to the BIP problem. In this example 20 changes from the current condition are permitted and the optimal objective function value is 336 (compared to the current condition's objective value of 289). This value of 20 changes is displayed on the slider control to indicate the maximum numbers of changes of x from current condition y, and the users can select their preferred minimum number of changes of x from y. Results for the two scenarios outlined here are further discussed below.

Scenario 1: Equal variable preferences

Under this scenario, the range of objective function value corresponding to different user-specified number of changes is displayed in Figure 13. Generally, the objective function value increases as the number of changes of solution x from the current condition y increases. However, as the user only specifies the lower bound on the number of changes required to the BIP problem, at times the actual number of changes from current condition y to the solution x do not equal the number of changes specified by the user. For instance, as seen in Figure 13, when the user specifies the number of changes to be in the range of five to seven, the objective function value is the same as that with user-specified number of changes of 8, that is, at $z = 316$. In other words, if the lower bound on the number of changes is specified to be a lower number than what is obtained, it may be due to ensure the feasibility of the solution to the model. In contrast, when the minimum number of changes is specified as five, it was not attractive for the model to increase the number of changes to nine although higher objective function value corresponding to nine changes was expected, which later can be explained further in Figure 14.

Another interesting insight gained from Figure 13 can be seen in the objective function value when comparing the difference between user-specified number of changes of (i) 17 and 18, and (ii) 15 and 16. In this case, as the number of changes increases there is a slight decrease in objective function value. This decrease can be interpreted as that it is not necessarily favorable to undergo the maximum number of changes (in this case 20), as the gain in objective value is not significant and the cost of changes may not offset the gain in the objective function value.

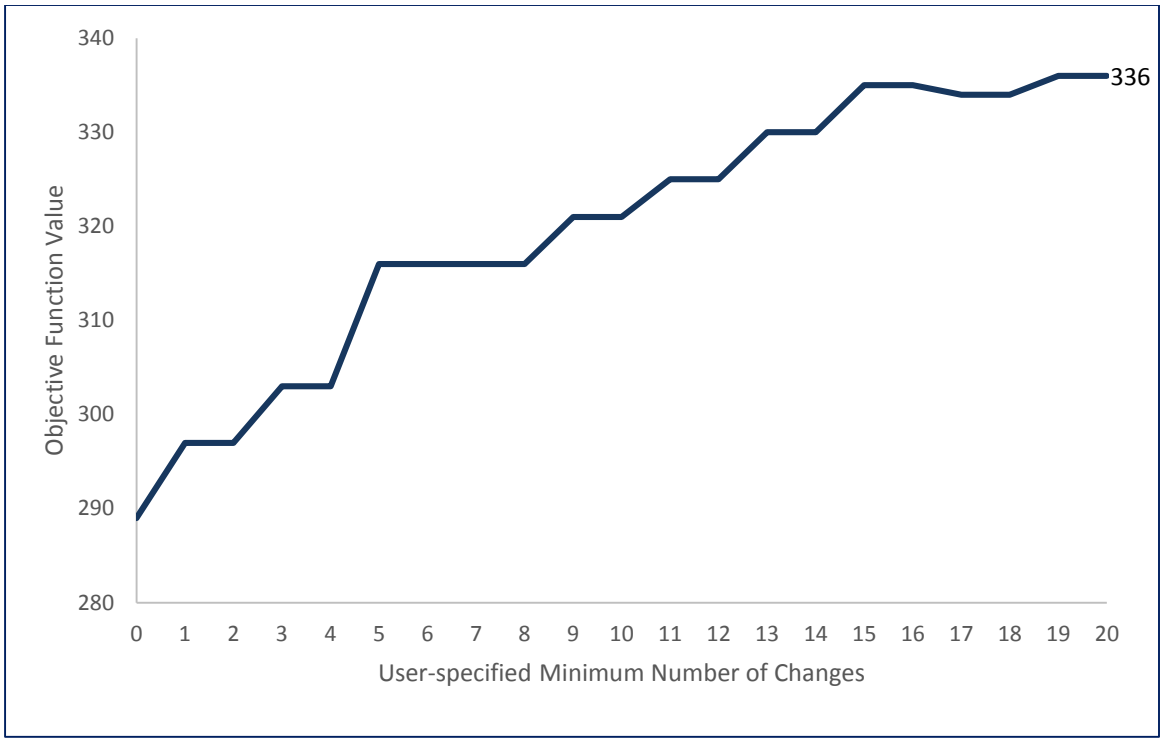


Figure 13: Objective Function Value Corresponding to User-specified Minimum Number of Changes of x_{ij} from y_{ij}

Figure 14 plots the $\lambda(x)$ value versus minimum number of changes of x from y defined by the user, where $\lambda(x)$ is defined as in equation (25). This figure illustrates that the incremental gain in the objective function value decreases as the number of changes increases. Additionally, Figure 14 explains the behavior in Figure 13. For instance, in Figure 13 the objective function value did not appear to increase when the user specified the number of changes to be five to eight. However, the modified objective function is to maximize ratio between solution quality and solution difference, which is $\max \lambda(x)$. Accordingly, in Figure 14, the decrease in the value of $\lambda(x)$ in the range of five to eight changes is a result of changes from the current condition (solution difference) only (the denominator ratio). Thus, our modified BIP is behaving as expected.

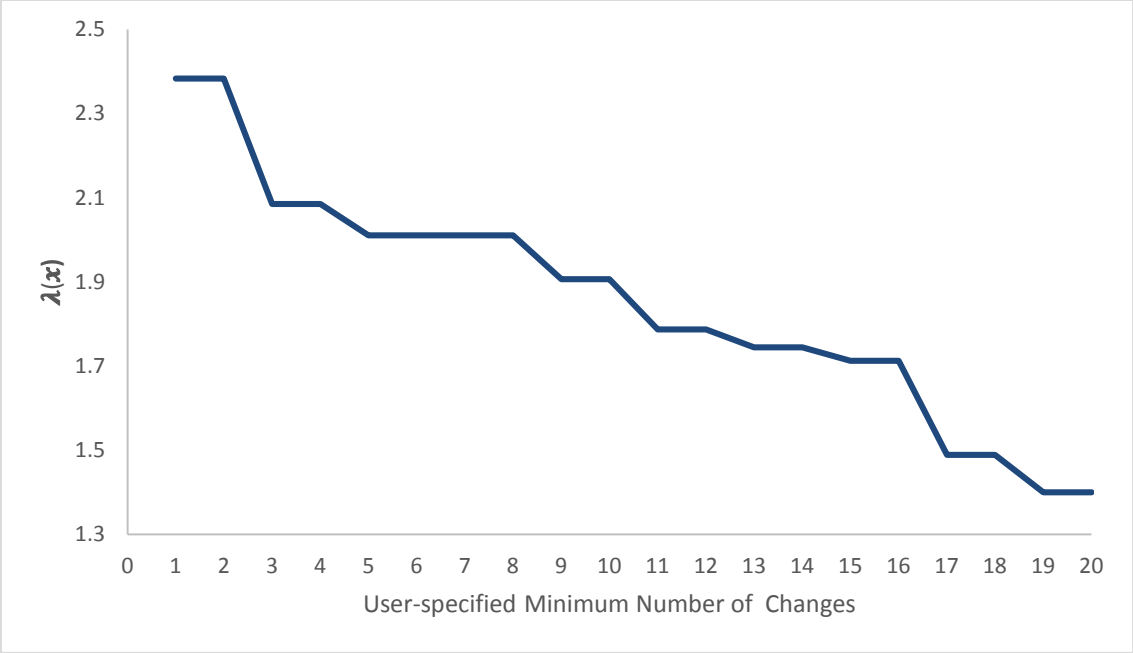


Figure 14: $\lambda(x)$ Value Corresponding to User-Define Minimum Number of Changes of x_{ij} from y_{ij}

Scenario 2: User-Specified Variable Preferences

In this scenario the user specifies variable preferences. Figure 15 illustrates the relationship between objective function values and user-specified number of changes. When the user- specified number of changes ranged from 2 to 10, the actual number of changes of the solution x from current condition y was always equal to 10, which produced the objective function value of $z = 316$. Similar to the first scenario, there was a decrease in the objective function value at a greater number of changes (14 to 15), indicating indicated that the increase in number of changes of x toward maximum number of changes was not favorable. The optimal objective function with 20 changes of x from y was found to be $z = 335$.

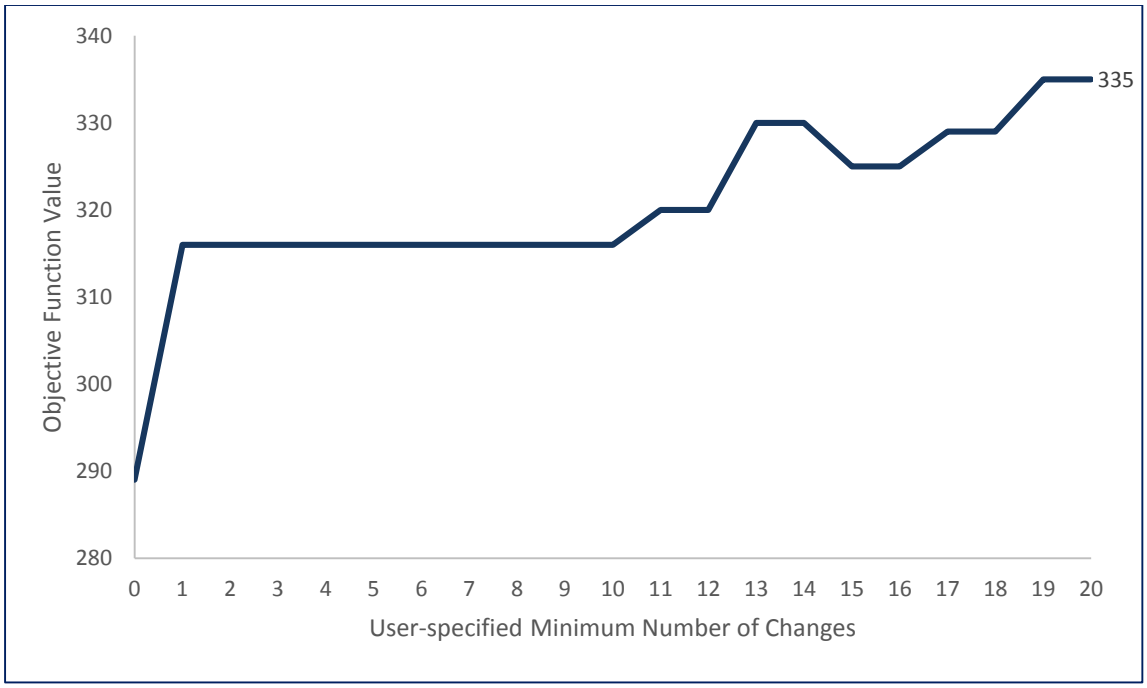


Figure 15: Objective Function Value Corresponding to User-specified Number of Changes of x_{ij} from y_{ij} with Variable Preference

Furthermore, the plot showing the value of $\lambda(x_{ij})$ is displayed in Figure 16. Similar to the first scenario, it can be seen that as number of changes incrementally increases, $\lambda(x_{ij})$ decreased as expected.

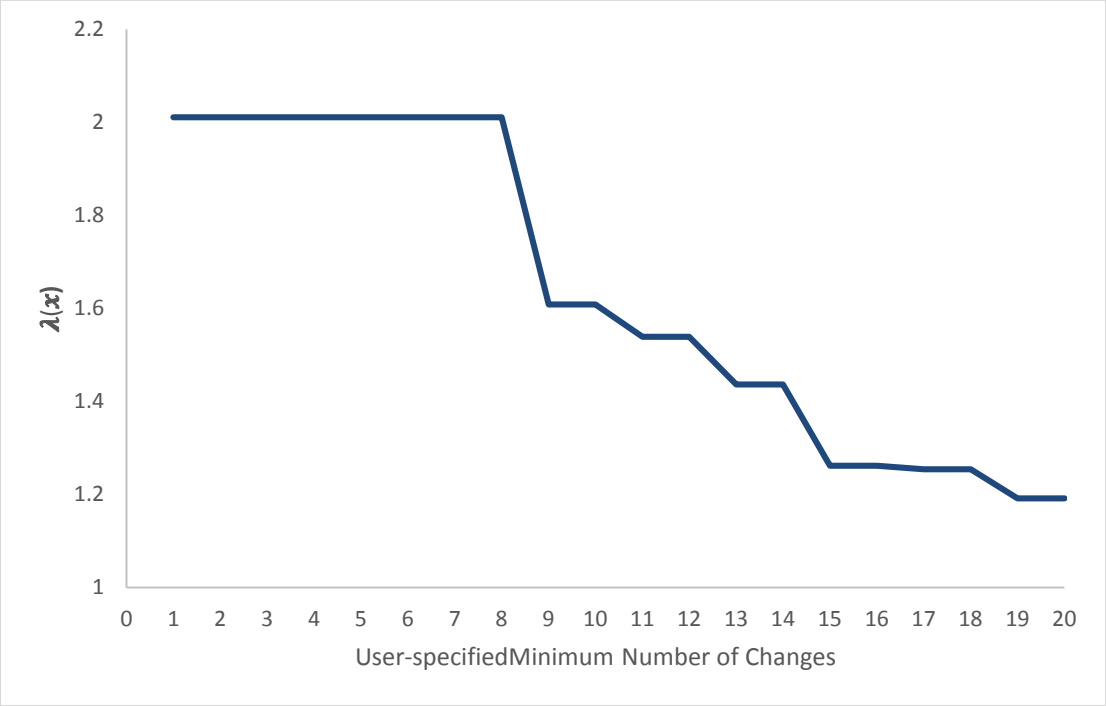


Figure 16: $\lambda(x)$ Value Corresponding to User-specified Minimum Number of Changes of x_{ij} from y_{ij} with Variable Preference

Comparison of the two variable preference scenarios

Although in both scenarios, the maximum number of changes from the current condition was set to 20, the optimal objective function value in the second scenario was less than that of the first scenario. This could be explained by the differences in variable preferences between the two scenarios. With a user-specified number of changes set to 20, Figure 17 and Figure 18 display the respective solutions to the user's BIP problem for both scenarios. It can be seen that different scenarios provide different outputs. For instance, x_{12} was equal to 1 in the first scenario, but equal to 0 in the second scenario. This can be explained by the assignment of a variable preference of x_{12}, which was set to three (Figure 12). Thus, it was not favorable for the model to change x_{12} from current condition, where $x_{12} = 0$, to 1 in the second scenario.

Machine\ Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0
2	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
4	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0

Figure 17: Optimal Solution to the BIP Problem without Variable Preferences (Scenario 1)

Machine\ Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	0	0	1	0	0	0	1	0	0	0	0	1	0
2	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
4	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0
5	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 18: Optimal Solution to the BIP Problem with Variable Preferences (Scenario 2)

Chapter 5. Conclusion and Recommendations

Solving optimization problems has long been a crucial research area for many disciplines, and many optimization problems featuring combinatorial structures can be modeled using binary integer variables. While there are rich practical applications of BIP models, existing exact solvers typically produce a single optimal solution to the BIP problems, which might be undesirable to the decision maker if it is too disruptive from the current condition. Thus, this project focused on developing an algorithm and a computational program capable of producing solutions that do not overly disrupt from the status quo, while still providing an improved objective function value to the BIP problems. Implemented with VBA and OpenSolver in an Excel environment, the program includes the following features:

(1) Provides step-by-step instructions;

(2) Reads in an existing model;

(3) Imports and validates user-specified input including current condition, objective coefficients, and variable preferences to the BIP problem;

(4) Allows the user to select desirable number of changes of solution from current condition;

(5) Solves the modified BIP model using Dinkelbach's algorithm based on user's preferences;

(6) Includes troubleshooting capabilities.

The user-interface guides the user through all aspects of the program from importing their models, to customizing the methods and receiving the desirable solutions.

For future work, it is recommended to enhance the program capability by allowing the solution of BIP models with a minimization objective. That said, it is possible for users to use the existing program by converting a minimization objective to maximization by simply multiplying through by -1. Furthermore, adapting the program to different solvers such as CPLEX and Gurobi would appeal to a larger audience.

Chapter 6. Industrial Engineering Reflection

To satisfy Accreditation Board for Engineering and Technology (ABET) related requirements, the Major Qualifying Project (MQP) in Industrial Engineering must represent certain capstone design experiences. According to ABET, the fundamental components of the design process should include the establishment of objectives and criteria, synthesis, analysis, construction, testing, and evaluation, which were fulfilled in this MQP. The objective of this project was to develop an algorithm and design a computational program to generate solution to BIP problem that not only improves the objective function value but also controls disruption from the current condition. First, we developed the model through literature review by understanding similar type of models and the approaches that were studied. From that, we proposed a detailed mathematical description that represents multiple objective functions including maximization in solution quality and minimization in disruption from the current condition. Then, we employed Dinkelbach's algorithm to solve for the desired solution through multiple iterations. Next, we synthesized the whole process into the Excel environment using VBA and OpenSolver and designed user-interface including detail instructions to assist users. Finally, we debugged and evaluated the program using several examples of BIP problems.

Furthermore, unlike other MQP groups, this was a single-student project with a disjointed project time frame. The MQP project had to be completed in two disjointed terms (i.e., 2/3 units in A term, 1/3 unit in C term); and as a result we experienced the discontinuity in the project progress. Most of the work was condensed at the beginning of the project, in A term. After the gap of B term, time was required to catch up with the previous progress, accelerate and finish the project. Additionally, with limited background in VBA, we had to familiarize ourselves with the environment, syntax, and debugging process. Another challenge is the limitation of OpenSolver documentation; most of our codes were leveraged from several online instructions and tutorials. Towards the end of the project, we had to test our program with several binary optimization problems, which helped us validate the functionality of our program.

As a result, the design process produced a relatively fast, accurate and functioning program. However, there were alternatives and constraints that were considered when designing the program. The time frame for the project only allowed for the program to account for the maximization cases in BIP problems. Moreover, as the program was created in Excel environment, there were limitations to the data set and models that it can solved. Thus future enhancements to the program would be to adapt the program to solve BIP minimization cases and extend the program to environments other than Excel.

Overall, the project was quite a process in learning and practice, when we had opportunities to experience a new programming environment relevant to binary optimization within an industrial discipline. The developing and debugging process enhanced our critical thinking and judgment skills. As we understood more about the users' needs, we developed a user-friendly interface to assist the decision makers in the optimizing process.

Bibliography

1. Aydin, E. M. & Fogarty, T., 2004. A Distributed Evolutionary Simulated Annealing Algorithm for Combinatorial Optimization Problems. *Journal of Heuristics,* March, 24(10), pp. 269-292.
2. Balas , E. & Jeroslow, R., 1972. Canonical Cuts on the Unit Hypercube. *SIAM J. Appl. Math,* July, Volume 23, pp. 61-69.
3. Battini, R., 1996. Reactive search: towards self- tuning heuristics. In: *Modern heuristic search methods.* s.l.:Wiley&Sons, pp. 61-83.
4. Bellman, R., 2003. *Dynamic Programming.* Mineola: Dover Publications.
5. Boland , N., Charkhgard , H. & Savelsbergh , M., 2013. *Criterion Space Search Algorithms for Biobjective Mixed 0-1 Integer Programming.* [Online]
   Available at: http://www.optimization-online.org/DB_FILE/2013/08/3987.pdf
   [Accessed 10 October 2013].
6. Borndörfer, R. & Grötschel, M., 2012. *Designing telecommunication networks by integer programming.* [Online]
   Available at: http://www.zib.de/groetschel/teaching/SS2012/120503Vorlesung-DesigningTelcomNetworks-reduced.pdf
   [Accessed 10 October 2013].
7. Boswell, R., 2012. *Exhautive Search Algorithm.* [Online]
   Available at: http://www.comp.rgu.ac.uk/staff/rab/CM3002/Notes/notes3.pdf
   [Accessed 7 September 2013].
8. Caramia, M. & Dell'Olmo, P., 2008. Multi-objective Optimization. In: *Multi-objective Management in Freight Logistics.* s.l.:Springer, pp. 11-36.

9. Cattrysse, D. G., Salomon, M. & Wassenhove, L. N. V., 2003. A set partitioning heuristic for the generalized assignment problem. *Elsevier,* 6 January, 72(1), pp. 167-174.
10. Chakrabarti, A., 2005. *Approximation Algorithms (continued).* [Online] Available at: http://www.cs.dartmouth.edu/~ac/Teach/CS105-Winter05/Notes/wan-ba-scribe.pdf [Accessed 10 September 2013].
11. Chinneck, J., 2004. *Practical Optimization: a Gentle Introduction.* s.l.:s.n.
12. Cormen , T. H., Leiserson , C. E., Rivest , R. L. & Stein , C., 2009. In: *Introduction to Algorithms.* s.l.:MIT Press, pp. 65-114.
13. Danna, E. & Woodruff, D., 2009. How to select a small set of diverse solutions to mixed integer programming problems. *Elsevier,* pp. 255-260.
14. Dantzig, G., 1951. Maximization of linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, pp. 339-347.
15. Dinkelbach, W., 1967. On Nonlinear Fractional Programming. *Management Science,* 1 March, 13(7), pp. 492-498.
16. Eberhart, R. C., Shi, Y. & Kenedy, J., 2001. *Swarm Intelligence (The Morgan Kaufmann Series in Evolutionary Computation).* s.l.:Academic Press.
17. Fonseca, Carlos, Fleming, Peter, 1993. *Genetic algorithms for multiobjective optimization: Formulation discussion.* s.l., Morgan Kaufmann Publishers Inc, pp. 416-423.
18. Fukuda, K. & Terlaky, T., 1997. *Criss-Cross Methods: A Fresh View on Pivot Algorithms.* [Online] Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.5739&rep=rep1&type=pdf [Accessed 15 December 2013].

19. Fylstra, D., Lasdon, L., Watson , J. & Waren , A., 1998. Design and use of the Microsoft Excel Solver. *Interfaces,* Volume 28, pp. 29-55.

20. Glover , F., Løkketangen , A. & Woodruff, D. L., 2000. Scatter search to generate diverse MIP solutions. In: *Computing Tools for Modeling, Optimization and Simulation, Interfaces in Computer Science and Operations Research.* s.l.:Kluwer Academic, pp. 299-317.

21. Gomes, C. P. & Williams, R., 2005. Approximation Algorithms. In: *Introduction to Optimization, Decision Support and Search Methodologies.* s.l.:Burke and Kendall, pp. 557-567.

22. Gurobi, 2014. *Gurobi optimizer reference manual.* [Online]
Available at: http://www.gurobi.com
[Accessed 10 March 2014].

23. Hochbaum, D., 2008. *Network Flows and Graphs.* [Online]
Available at: http://www.ieor.berkeley.edu/~ieor266/Lecture1-5.pdf
[Accessed 6 November 2013].

24. Huang, C.-Y. & Wang, T.-C., 2011. *Gradient Methods for Binary Integer Programming.* Los Angeles, s.n., pp. 410-416.

25. Hwang, Ching-Lai, Masud, Abu Syed Md , 1979. *Multiple objective decision making, methods and applications: a state-of-the-art survey.* s.l.:Springer-Verlag.

26. IBM, 2014. *CPLEX Optimizer.* [Online]
Available at: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
[Accessed 10 March 2014].

27. Karp, R. M., 2010. Reducibility Among Combinatorial Problems. In: *50 Years of Integer Programming 1958-2008.* s.l.:Springer Berlin Heidelberg, pp. 219-241.

28. Klein, P. & Young, N., 1999. Chapter 34. Algorithms and Theory of Computation Handbook. In: *Approximation algorithms for NP-hard optimization problems.* s.l.:CRC Press.

29. Kokash, N., 2005. *An introduction to heuristic algorithms,* Trento: s.n.

30. Konak, A., Coit, D. W. & Smith, A. E., 2006. Multi-objective optimization using genetic algorithms: A tutorial. *Elsevier,* 91(9), p. 992–1007.

31. Lougee-Heimer, R., 2010. The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development,* 06 April.pp. 57-66.

32. Mathworks, 2014. *Optimization Toolbox.* [Online]
Available at: http://www.mathworks.com/products/optimization/
[Accessed 10 March 2014].

33. Matoušek, Jirí, Gärtner, Bernd , 2007. Maximum-Weight Matching. In: *Understanding and Using Linear Programming.* Heidelberg: Springer, pp. 31-35.

34. Microsoft, 2013. *VBA for Office developers.* [Online]
Available at: http://msdn.microsoft.com/en-us/office/ff688774.aspx
[Accessed 14 December 2013].

35. Murthy, S. et al., 1999. Cooperative multiobjective decision support for the paper industry. *Interfaces 29,* pp. 5-30.

36. Nemhauser, G. & Wolsey, L., 1998. *Integer and Combinatorial Optimization.* s.l.:John Wiley & Sons, Inc..

37. Nguyen, Huy, Onak, Krzysztof, 2008. *Constant-Time Approximation Algorithms via Local Improvements.* s.l., The 49th Annual Symposium on Foundations of Computer Science.

38. Roditty , L. & Shapira, A., 2011. All-Pairs Shortest Paths with a Sublinear Additive Error. *Algorithmica,* pp. 621-633.

39. Schaible, S., 1976. Fractional Programming. II, on Dinkelbach's Algorithm. *Management Science,* April, 22(8), pp. 868-873.

40. Serang, O., 2012. Conic Sampling: An Efficient Method for Solving Linear and Quadratic Programming by Randomly Linking Constraints within the Interior. *PLoS ONE 7(8): e43706.*

41. Stutzle, D.-W. T. G., 1998. *Local Search Algorithms for Combinatorial Problem,* s.l.: s.n.

42. Thai, M., 2013. *Approximation Algorithms:LP Relaxation, Rounding, and Randomized Rounding Techniques.* [Online]
Available at: http://www.cise.ufl.edu/class/cot5442sp13/Notes/Rounding.pdf
[Accessed 15 September 2013].

43. Trapp, A. & Konrad, R., 2013. *Finding diverse solutions of high quality to binary integer programs,* s.l.: Optimization Online.

44. Turton, R. et al., 2012. *Analysis, Synthesis, and Design of Chemical Processes.* Forth Edition ed. s.l.:Prentice Hall.

Appendix A

Initialize Button

```
Private Sub Initialize_Click()
   'Check existence
   Dim ws As Worksheet
   On Error Resume Next
      Set ws = Sheets("Model")
   On Error GoTo 0
   If ws Is Nothing Then
      MsgBox "Warning: Model worksheet doesn't exist"
      Exit Sub
   End If

   Sheets("Model").Activate
   lastrow = ws.UsedRange.rows.Count   'last row of data
   a = lastrow + 100 ' end of file

   'Open csv file
   temp1 = OpenFile(ActiveWorkbook.Path & "\user_initial.csv", a + 1, 1)
   temp2 = OpenFile(ActiveWorkbook.Path & "\user_weight.csv", a + 2, 2)
   temp3 = OpenFile(ActiveWorkbook.Path & "\user_objective.csv", a + 3, 3)

   If temp1 = -1 Or temp2 = -1 Or temp3 = -1 Then
      ws.Range(ws.Cells(a + 1, 1), ws.Cells(a + 12, 1)).EntireRow.ClearContents
      Exit Sub
   End If

   If temp1 = 0 Or temp3 = 0 Then
      MsgBox "Warning: Missing input files"
      ws.Range(ws.Cells(a + 1, 1), ws.Cells(a + 12, 1)).EntireRow.ClearContents
      Sheets("Main").Activate
      Exit Sub
   End If

   'No weight specified
   If temp2 = 0 Then
      temp2 = temp1
      ws.Cells(a + 2, 1).Resize(1, temp1) = 1
   End If

   'Cell locations
   Initial = ws.Cells(a + 1, 1).Resize(1, temp1).Address()
   Variable = ws.Cells(a + 5, 1).Resize(1, temp1).Address()
```

```
    'Solver
    typ = 1   '1: OpenSolver, other: Built-in Solver
    State = SolverGet(TypeNum:=1) 'Check solver
    If IsError(State) Then
        MsgBox "Warning: You have not used Solver on the active sheet"
        ws.Range(ws.Cells(a + 1, 1), ws.Cells(a + 12, 1)).EntireRow.ClearContents
        Sheets("Main").Activate
        Exit Sub
    End If

    SolverSave ("Main!P600:P1000")

    'Variable cell location
    orgVariable = SolverGet(TypeNum:=4) 'Get variable location as string
    vRow = ws.Range(orgVariable).rows.Count
    vCol = ws.Range(orgVariable).Columns.Count
    ltemp = vRow * vCol

    If temp1 <> ltemp Or temp2 <> ltemp Or temp3 <> ltemp Then
        MsgBox "Warning: Numbers of inputs are not consistent" & vbNewLine & "Exit to main worksheet"
        ws.Range(ws.Cells(a + 1, 1), ws.Cells(a + 12, 1)).EntireRow.ClearContents
        Sheets("Main").Activate
        Exit Sub
    End If

    'Variable to new cell
    For ii = 1 To vRow
        ws.Range(ws.Cells(a + 5, 1 + vCol * (ii - 1)), ws.Cells(a + 5, vCol * ii)).FormulaArray = "=" &
ws.Range(orgVariable).rows(ii).Address()
    Next ii

    'Optimize
    RunSolver (typ)
    dx_cell = Cells(a + 12, 6).Address()
    ws.Range(dx_cell).Formula = "=SUMIF(" & Initial & ",0," & Variable & ")-SUMIF(" & Initial & ",1," &
Variable & ")+SUM(" & Initial & ")"

    'Set scrollbar
    If ws.Range(dx_cell) = 0 Then   'when max changes = 0, initial solution is already optimal
        MsgBox "Your initial solution is already optimal. Any changes to the solution may decrease the
objective value"
        ScrollBar1.Min = 0
        ScrollBar1.Max = 0
        Label1.Caption = ScrollBar1.Value
    Else
        ScrollBar1.Min = 1
        ScrollBar1.Max = ws.Range(dx_cell)
        Label1.Caption = ScrollBar1.Value
```

```
    End If

    'Feasibility
    SolverAdd CellRef:=orgVariable, relation:=2, FormulaText:=Initial
    ic = SolverSolve(True)
    SolverDelete CellRef:=orgVariable, relation:=2
    If ic > 2 Then
        MsgBox "Warning: Infeasible initial condition"
        ws.Range(ws.Cells(a + 1, 1), ws.Cells(a + 12, 1)).EntireRow.ClearContents
        Sheets("Main").Activate
        Exit Sub
    End If

    '''
    ws.Range(ws.Cells(a + 1, 1), ws.Cells(a + 12, 1)).EntireRow.ClearContents
    ws.Range("A1").Select
    Sheets("Main").Activate
End Sub
```

Solve Button

```
Private Sub Solve_Click()
   'Changes
   changes = ScrollBar1.Value
   If changes = 0 Then
      MsgBox "Warning: There must be at least ONE change"
      Exit Sub
   End If

   'Last row
   If a = 0 Then
      MsgBox "Warning: Click Initialize first"
      Exit Sub
   End If

   'Maxloop
   maxloop = Range("P48")
   If Not IsNumeric(maxloop) Then
      Range("P48") = 100
   ElseIf CInt(maxloop) < 1 Then
      Range("P48") = 100
   End If
   maxloop = Range("P48")

   'Check existence
   Dim ws As Worksheet
   On Error Resume Next
      Set ws = Sheets("Model")
   On Error GoTo 0
   If ws Is Nothing Then
      MsgBox "Warning: Model worksheet doesn't exist"
      Exit Sub
   End If

   ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
   Sheets("Model ").Activate

   'Solver
   typ = 1   '1: OpenSolver, other: Built-in Solver
   State = SolverGet(TypeNum:=1) 'Check solver
   If IsError(State) Then
      MsgBox "Warning: You have not used Solver on the active sheet" & vbNewLine & "Exit to main
worksheet"
      Sheets("Main").Activate
      Exit Sub
   End If
```

```
    'Open csv file
    temp1 = OpenFile(ActiveWorkbook.Path & "\user_initial.csv", a + 1, 1)
    temp2 = OpenFile(ActiveWorkbook.Path & "\user_weight.csv", a + 2, 2)
    temp3 = OpenFile(ActiveWorkbook.Path & "\user_objective.csv", a + 3, 3)

    'No weight specified
    If temp2 = 0 Then
        temp2 = temp1
        ws.Cells(a + 2, 1).Resize(1, temp1) = 1
    End If

    'Cell locations
    Initial = ws.Cells(a + 1, 1).Resize(1, temp1).Address()
    weight = ws.Cells(a + 2, 1).Resize(1, temp1).Address()
    Objective = ws.Cells(a + 3, 1).Resize(1, temp1).Address()
    wVariable = ws.Cells(a + 4, 1).Resize(1, temp1).Address()
    Variable = ws.Cells(a + 5, 1).Resize(1, temp1).Address()

    'Variable cell location
    orgVariable = SolverGet(TypeNum:=4) 'Get variable location as string
    vRow = ws.Range(orgVariable).rows.Count
    vCol = ws.Range(orgVariable).Columns.Count

    'Variable to new cell
    For ii = 1 To vRow
        ws.Range(ws.Cells(a + 5, 1 + vCol * (ii - 1)), ws.Cells(a + 5, vCol * ii)).FormulaArray = "=" &
ws.Range(orgVariable).rows(ii).Address()
    Next ii

    ws.Range(wVariable).FormulaArray = "=" & Variable & "*" & weight

   'Initialize Dinkelbach 1
    nx_cell = ws.Cells(a + 12, 3).Address()
    ws.Range(nx_cell).Formula = "= SUMPRODUCT(" & Objective & "," & Variable & ")-SUMPRODUCT(" &
Objective & "," & Initial & ")"
    dxw_cell = ws.Cells(a + 12, 4).Address()
    ws.Range(dxw_cell).Formula = "=SUMIF(" & Initial & ",0," & wVariable & ")-SUMIF(" & Initial & ",1," &
wVariable & ")+SUMPRODUCT(" & weight & "," & Initial & ")"

    'Find q(x)
    temp_cell = ws.Cells(a + 11, 1).Address()
    ws.Range(temp_cell).Formula = "=SUMPRODUCT(" & Objective & "," & Variable & ")-SUMPRODUCT("
& Objective & "," & Initial & ")"
    SolverOK setCell:=temp_cell, MaxMinVal:=1, ByChange:=orgVariable 'Objective function to find max
q(x)for normalization factor
    SolverAdd CellRef:=temp_cell, relation:=3, FormulaText:="1e-6" 'Constraint q(x)>=0 to have cx-
cy>=1e-6
```

```
    RunSolver (typ)

    'Save and clear
    qx = ws.Range(temp_cell).Value
    Q1 = ws.Range(nx_cell) / ws.Range(dxw_cell)
    SolverDelete CellRef:=temp_cell, relation:=3
    ws.Range(temp_cell).ClearContents

    'Find s(x)
    ws.Range(temp_cell).Formula = "=SUMIF(" & Initial & ",0," & wVariable & ")-SUMIF(" & Initial & ",1,"
& wVariable & ")+SUMPRODUCT(" & weight & "," & Initial & ")"
    SolverOK setCell:=temp_cell, MaxMinVal:=1, ByChange:=orgVariable 'Find max of s(x)for
normalization factor
    SolverAdd CellRef:=temp_cell, relation:=3, FormulaText:="1" 'Constraint s(x)>=1
    RunSolver (typ)

    'Save and clear
    sx = ws.Range(temp_cell).Value
    Q2 = ws.Range(nx_cell) / ws.Range(dxw_cell)
    SolverDelete CellRef:=temp_cell, relation:=3
    ws.Range(temp_cell).ClearContents

    'Normalize factor
    norm_cell = ws.Cells(a + 12, 1).Address()
    ws.Range(norm_cell) = sx / qx

    'Initialize Dinkelbach 2
    lambda_cell = ws.Cells(a + 12, 2).Address()
    'ws.Range(lambda_cell) = 0
    ws.Range(lambda_cell) = Application.Max(Q1, Q2, 0)
    nxqdx_cell = ws.Cells(a + 12, 5).Address()
    ws.Range(nxqdx_cell).Formula = "=" & norm_cell & "*" & nx_cell & "-" & lambda_cell & "*" &
dxw_cell
    dx_cell = Cells(a + 12, 6).Address()
    ws.Range(dx_cell).Formula = "=SUMIF(" & Initial & ",0," & Variable & ")-SUMIF(" & Initial & ",1," &
Variable & ")+SUM(" & Initial & ")"

    'Dinkelbach
    SolverOK setCell:=nxqdx_cell, MaxMinVal:=1, ByChange:=orgVariable 'Define objective function: max
    SolverAdd CellRef:=nx_cell, relation:=3, FormulaText:="0" 'constraint N(x)>=0
    SolverAdd CellRef:=dx_cell, relation:=3, FormulaText:=CStr(changes)  'constraint D(x)>=user_define??

    'Loop until zero
    cnt = 0
    While Abs(ws.Range(nxqdx_cell)) > 0.000001
        cnt = cnt + 1
        If cnt > maxloop Then
            SolverDelete CellRef:=nx_cell, relation:=3
```

```
      SolverDelete CellRef:=dx_cell, relation:=3
      SolverOK setCell:=State, MaxMinVal:=1, ByChange:=orgVariable
      ws.Range(ws.Cells(a + 1, 1), ws.Cells(a + 12, 1)).EntireRow.ClearContents
      ws.Range("A1").Select
      MsgBox "Warning: Number of iterations exceeds the allowed"
      Sheets("Main").Activate
      Exit Sub
    End If
    RunSolver (typ)

    'Update q
    ws.Range(lambda_cell) = ws.Range(norm_cell) * ws.Range(nx_cell) / ws.Range(dxw_cell)
  Wend

  'Old Objective
  old_cell = ws.Cells(a + 12, 7).Address()
  ws.Range(old_cell).Formula = "=SUMPRODUCT(" & Objective & "," & Initial & ")"
  oldobj = ws.Range(old_cell)

  'Save
  achanges = ws.Range(dx_cell)
  newobj = ws.Range(State)

  'Clear
  SolverDelete CellRef:=nx_cell, relation:=3
  SolverDelete CellRef:=dx_cell, relation:=3
  SolverOK setCell:=State, MaxMinVal:=1, ByChange:=orgVariable
  ws.Range(ws.Cells(a + 1, 1), ws.Cells(a + 12, 1)).EntireRow.ClearContents
  ws.Range("A1").Select

  ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

  MsgBox "Solution found!" & vbNewLine & _
  "Objective value is " & CStr(newobj) & vbNewLine & _
  "New objective value improves over the initial objective value of " & _
  CStr(oldobj) & " by " & CStr(newobj - oldobj) & vbNewLine & _
  "Number of changes is " & CStr(achanges) & vbNewLine & _
  "Number of iterations is " & CStr(cnt)
End Sub
```

Reset Button

```
Private Sub Reset_Click()
   ScrollBar1.Min = 0
   ScrollBar1.Max = 0

   'Check existence
   Dim ws As Worksheet
   On Error Resume Next
      Set ws = Sheets("Model ")
   On Error GoTo 0
   If ws Is Nothing Then
      MsgBox "Warning: Model worksheet doesn't exist"
      Exit Sub
   End If

   Sheets("Model ").Activate

   SolverReset
   SolverLoad ("Main!P600:P1000")

      Sheets("Main").Activate

End Sub
```

Miscellaneous

```
Private Function OpenFile(filepath, rows, inputfile)
  'Open file and split cells
  Open filepath For Input As #1
  Dim LineFromFile As String
  Do Until EOF(1)
    Line Input #1, temp
    LineFromFile = LineFromFile & "," & temp
  Loop
  Close #1
  LineItems = Split(LineFromFile, ",")

  'Check input data and Remove null element
  Dim i, j As Integer
  ReDim newArr(LBound(LineItems) To UBound(LineItems))
  For i = LBound(LineItems) To UBound(LineItems)
    If LineItems(i) <> "" Then
      Select Case inputfile
        Case 1
          If LineItems(i) = 0 Or LineItems(i) = 1 Then
            newArr(j) = CInt(LineItems(i))
            j = j + 1
          Else
            MsgBox "Warning: Input file user_initial.csv has wrong data type."
            OpenFile = -1
            Exit Function
          End If
        Case 2
          If IsNumeric(LineItems(i)) And CInt(LineItems(i)) >= 0 Then
            newArr(j) = CInt(LineItems(i))
            j = j + 1
          Else
            MsgBox "Warning: Input file user_weight.csv has wrong data type."
            OpenFile = -1
            Exit Function
          End If
        Case 3
          If IsNumeric(LineItems(i)) Then
            newArr(j) = CInt(LineItems(i))
            j = j + 1
          Else
            MsgBox "Warning: Input file user_objective.csv has wrong data type."
            OpenFile = -1
            Exit Function
          End If
        End Select
```

```
      End If
   Next
   ReDim Preserve newArr(LBound(LineItems) To j)

   'Return
   Sheets("Model ").Cells(rows, 1).Resize(1, UBound(newArr) - LBound(newArr)) = newArr  'store array in
specified row
   OpenFile = UBound(newArr) - LBound(newArr)   'return length of array
End Function

Private Sub RunSolver(typ)
   If typ = 1 Then
      RunOpenSolver False
   Else
      SolverSolve True
   End If
End Sub


Private Sub ScrollBar1_Change()
   Label1.Caption = ScrollBar1.Value
End Sub
```