Approximation in Multiobjective Optimization with Applications

Lakmali Weerasena
Clemson University, lweeras@clemson.edu

Approximation in Multiobjective Optimization with Applications

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Mathematical Sciences

by
Lakmali Weerasena
December 2013

Abstract

Over the last couple of decades, the field of multiobjective optimization has received much attention in solving real-life optimization problems in science, engineering, economics and other fields where optimal decisions need to be made in the presence of trade-offs between two or more conflicting objective functions. The conflicting nature of objective functions implies a solution set for a multiobjective optimization problem. Obtaining this set is difficult for many reasons, and a variety of approaches for approximating it either partially or entirely have been proposed.

In response to the growing interest in approximation, this research investigates developing a theory and methodology for representing and approximating solution sets of multiobjective optimization problems. The concept of the tolerance function is proposed as a tool for modeling representation quality. Two types of subsets of the set being represented, covers and approximations, are defined, and their properties are examined.

In addition, approximating the solution set of the multiobjective set covering problem (MOSCP), one of the challenging combinatorial optimization problems that has seen limited study, is investigated. Two algorithms are proposed for approximating the solution set of the MOSCP, and their approximation quality is derived. A heuristic algorithm is also proposed to approximate the solution set of the MOSCP. The performance of each algorithm is evaluated using test problems. Since the MOSCP has many real-life applications, and in particular designing reserve systems for ecological species is a common field for its applications, two optimization models are proposed in this dissertation for preserving reserve sites for species and their natural habitats.

Acknowledgments

I would like to express my heartfelt gratitude to my advisor, Dr. Margaret M. Wiecek, for her guidance and support. Working with her these past three years has been challenging as well as rewarding. Her insights have improved not only this dissertation but also my understanding of and ability to conduct research. I admire her expertise and am grateful that she was so willing to share it with me.

My committee members, Dr. Pietro Belotti, Dr. Mary Elizabeth Kurz, Dr. Beth Novick and Dr. Matthew Saltzman, each an expert in his/her respective field, were reliable sources for additional help and suggestions along the way. I am fortunate to have received their time and commitment during my fourth exam, which resulted in new ideas for this dissertation, and their numerous suggestions and ideas for further improvement and additional research directions for my work.

A significant impact on my research is also credited to Dr. Douglas Shier, my master's research advisor, who deserves many thanks because his commitment and fascination have also broadened my general interests in various aspects of this research. I would also like to thank Dr. Banu Soylu, Dr. David Tonkyn, and Dr. Daniel Vanderpooten for sharing their experiences, ideas and valuable time with me to successfully complete the manuscripts included in this dissertation.

Special thanks go to my family, especially to my mother Mrs. Seetha Weerasena; without her encouragement and support, I would not have been able to be so successful in my academic career. I am also grateful for my father and grandparents; although they are no longer here to see my success, I hope they know they are always in my heart.

My warmest thanks belong to my husband, Dr. Damitha Bandara, for generously showing his devotion and creating new wonderful moments in my life. His overwhelming support and encouragement during these last months mean more to me than he can imagine.

While my supporters are too numerous to mention, I also thank all my other teachers and the faculty who worked with me, in particular Dr. W. B. Daundasekara, Dr. Shelton Perera, Dr.Chris Cox, Dr. K. B. Kulasekera, and Dr. Robert Taylor for supporting me in various ways. Finally, I would like to thank Mrs. Barbara Ramirez, Mr. Nathanael Black, Dr. Brian Dandurand, and all of my friends who helped me to complete this dissertation successfully.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Life is about decisions. No matter if made by a group or an individual, decisions usually involve several conflicting objectives based on group's or individual goals and preferences. The presence of several objectives usually does not allow decision makers to identify a universal best decision, meaning they need to make a choice among all possible alternatives to reach the best decision. If it is too difficult to find all possible decisions or even their subset, potential alternatives are identified and decision makers select the best among them. These ideas form the heart of this dissertation.

For further guidance, a brief outline of the general organization of Chapter 1 follows. Section 1.1 begins with a review of various relevant mathematical concepts, the current literature, solution approaches to decision problems in the presence of multiple objectives, and a specific type of the decision problem and its application. A statement of the research objectives is included in Section 1.2, while the research contributions for each specific goal are found in Section 1.3. The content of Chapters 2 through 5 is summarized in Section 1.4, with concluding comments together with ideas and research directions for possible future work being presented in Section 1.5.

1.1 State of the art

Decision making in the presence of multiple conflicting criteria motivates the posing of mathematical optimization problems for which a finite number of objectives is represented by a vector-valued function, thereby giving rise to a multiobjective optimization problem (MOP). Due to the conflicting nature of criteria, a solution that is optimal with respect to all of the objective functions simultaneously may not exist. Instead, an optimal solution set is implied whose elements are characterized by the inability to improve any of these solutions with respect to any objective function without deteriorating at least one other objective function. The success of applying multiobjective optimization in practice depends, among other things, on the ability to compute the elements of the optimal solution set. This set is typically large, and it is often difficult or even impossible to obtain its exact description.

To extend the current theories of multiobjective optimization, which involves the characterization of the solution sets and the development of applicable methods for their generation, this section begins with a review of various relevant mathematical concepts and of the pertinent literature. Section 1.1.1 introduces basic concepts and notations, while multiobjective optimization and the corresponding concepts of efficiency are reviewed in Section 1.1.2. Approaches to generating solution sets of MOPs are grouped into two categories and discussed in Section 1.1.3. More specifically, Section 1.1.4 introduces the multiobjective set covering problem and discusses solution generating approaches for it. Designing reserve systems for protecting species, a common field for its applications, is discussed in Section 1.1.5.

1.1.1 Basic concepts and notations

This section discusses some relevant mathematical concepts and notations related to multiobjective optimization that are used extensively throughout this dissertation. Let n and p be two positive integers and let $\mathcal{R}^n$ and $\mathcal{R}^p$ be two Euclidean spaces. Let $X \subseteq \mathcal{R}^n$

be a nonempty set. A function

$$f : X \to \mathcal{R}^p$$

is a mapping that assigns to each element x in the domain X a unique element $y \in \mathcal{R}^P$ denoted by $y = f(x)$. The image of X under f is denoted by

$$Y := f(X) := \{y \in \mathcal{R}^p : y = f(x) \text{ for some } x \in X\}. \tag{1.1}$$

If $p = 1$, then the function f is a scalar-valued function. Otherwise it is a vector-valued function denoted by $f = (f_1, f_2, \ldots, f_p)$, where each $f_i : X \to \mathcal{R}$ is a scalar-valued function. For a vector valued function $f = (f_1, f_2, \ldots, f_p)$, we write

$$f(x^1) \leqq f(x^2) \text{ if and only if } f_i(x^1) \leq f_i(x^2) \text{ for all } i = 1, 2, \ldots, p \tag{1.2a}$$

$$f(x^1) \leq f(x^2) \text{ if and only if } f_i(x^1) \leqq f_i(x^2) \text{ and } f(x^1) \neq f(x^2) \tag{1.2b}$$

$$f(x^1) < f(x^2) \text{ if and only if } f_i(x^1) < f_i(x^2) \text{ for all } i = 1, 2, \ldots, p \tag{1.2c}$$

and use $\geqq, \geq$ and $>$ accordingly. Binary relations and partial orders play an important role in multiobjective optimization. We define the notion of a binary relation on an arbitrary set S as follows:

Definition 1.1.1. *A* binary relation R *on a set* S *is a subset of the Cartesian product* $S \times S$. *A binary relation is said to be*

(i) reflexive *if* $(s, s) \in R$ *for all* $s \in S$,

(ii) irreflexive *if* $(s, s) \notin R$ *for all* $s \in S$,

(iii) symmetric *if* $(s^1, s^2) \in R \Leftrightarrow (s^2, s^1) \in R$ *for all* $s^1, s^2 \in S$,

(iv) asymmetric *if* $(s^1, s^2) \in R \Rightarrow (s^2, s^1) \notin R$ *for all* $s^1, s^2 \in S$,

(v) antisymmetric *if* $(s^1, s^2) \in R$ *and* $(s^2, s^1) \in R \Rightarrow s^1 = s^2$ *for all* $s^1, s^2 \in S$,

(vi) transitive *if* $(s^1, s^2) \in R$ *and* $(s^2, s^3) \in R \Rightarrow (s^1, s^3) \in R$ *for all* $s^1, s^2, s^3 \in S$.

Definition 1.1.2. *A binary relation $R \subseteq S \times S$ on a set S is called*

(i) *a* preorder *if it is reflexive and transitive*

(ii) *a* partial order *if it is reflexive, transitive, and antisymmetric*

(iii) *a* strict partial *order if it is irreflexive and transitive.*

In this thesis, we use the concept of binary relations to introduce orders onto the set $Y \subset \mathcal{R}^p$, and for convenience we define these orders on the complete Euclidean space $\mathcal{R}^p$.

Remark 1.1.1. *The binary relations $\geqq$ and $\leqq$ in (1.2) are preorders on $\mathcal{R}^p$, and the binary relations $\geq$ and $\leq$ are strict partial orders. Furthermore, the binary relations $>$ and $<$ are also strict partial orders.*

Throughout this thesis, we use several types of cones which we define now.

Definition 1.1.3. *A set $C \in \mathcal{R}^p$ is called a* cone *if $\lambda C \subseteq C$ for all $\lambda > 0$. A cone C is* convex *if and only if $C + C \subseteq C$. Further, a cone C is said to be* pointed *if $\sum_{i=1}^{k} c^i = 0$ if and only if $c^i = 0$ for all $c^i \in C,\ i = 1, 2, \ldots, k$.*

Remark 1.1.2. *According to Definition 1.1.3 a cone may contain the origin or not ([90], [91]).*

Definition 1.1.4. *Let C be a cone in $\mathcal{R}^p$ and v be a vector in $\mathcal{R}^p$. A cone with vertex v is defined as a* translation *$v + C$ of the cone C.*

Definition 1.1.5. *A set $C \in \mathcal{R}^p$ is called a* polyhedral set *if there exists a matrix $A \in \mathcal{R}^{l \times p}$ and a vector $b \in \mathcal{R}^l$ so that*

$$C = C(A, b) := \{y \in \mathcal{R}^p : Ay \geqq b\}.$$

If $b = 0$, then

$$C = C(A, 0) := \{y \in \mathcal{R}^p : Ay \geqq 0\}$$

is called a polyhedral cone*.*

The nonnegative, nonzero, and positive orthant of $\mathcal{R}^p$

$$\mathcal{R}^p_{\geqq} := \{y \in \mathcal{R}^p : y \geqq 0\} \tag{1.3a}$$

$$\mathcal{R}^p_{\geq} := \{y \in \mathcal{R}^p : y \geq 0\} \tag{1.3b}$$

$$\mathcal{R}^p_{>} := \{y \in \mathcal{R}^p : y > 0\} \tag{1.3c}$$

are pointed convex cones and play an important role in the following discussion of multiobjective optimization. We call these cones Pareto cones.

The definition of orders discussed above may be given in terms of cones when they are used to define cone-relations between the elements of the set Y [110].

Definition 1.1.6. *Let $y^1, y^2 \in Y$ and C be a cone. A cone relation is defined as*

$$y^1 \leqq_C y^2 \text{ if and only if } y^2 - y^1 \in C \tag{1.4a}$$

$$y^1 \leq_C y^2 \text{ if and only if } y^2 - y^1 \in C \setminus \{0\} \tag{1.4b}$$

Equivalently, the relation $y^1 \leqq_C y^2$ implies that there exists $d \in C$ such that $d = y^2 - y^1 \in C$ and the relation $y^1 \leq_C y^2$ implies that there exists $d \in C$, $d \neq 0$ such that $d = y^2 - y^1 \in C$.

We now introduce the concepts of dominated points, nondominated points and nondominated sets as established by Yu in 1974 [110] using the cone relation $\leq_C$ given in Definition 2.2.1.

Definition 1.1.7. *Let $Y \in \mathcal{R}^p$ be a nonempty set and C be a cone in $\mathcal{R}^p$. A point $y' \in Y$ is called a dominated point of the set Y with respect to the cone Y if there exists a point $y \in Y$ such that $y \leq_C y'$.*

Definition 1.1.8. *A point $y' \in Y$ is called a nondominated point of the set Y with respect to the cone C if there does not exist $y \in Y$ and $d \in C$, $d \neq 0$, such that $y' = y + d$ or,* *equivalently, there does not exist $y \in Y$ such that $y \leq_C y'$. The set of all nondominated points of Y with respect to the cone C is denoted by $N(Y, C)$.*

1.1.2 Multiobjective optimization

While the optimization of a scalar-valued function is understood in terms of minimization or maximization, the optimization of a vector-valued function requires the introduction of a different concept of optimality. The optimization of a scalar-valued function (also called an objective function) is well defined based on the order of real numbers. However, this concept is not well defined for a vector-valued function.

The concept of partial orders introduced in the previous section is used here to define optimality in multiobjective optimization. We assume that each scalar-valued function $f_i, i = 1, 2, \ldots, p$, of the vector-valued function f is to be minimized. Let $X \subseteq \mathcal{R}^n$ be a nonempty set and $f : X \to \mathcal{R}^p$ be a vector-valued function. A multiobjective optimization problem (MOP) is defined as

$$\begin{aligned} &\min\ f(x) \\ &\text{s.t. } x \in X \end{aligned} \tag{1.5}$$

and is denoted (X, f).

Throughout this dissertation it is assumed that the objective function f maps the set of feasible decisions X from the decision space $\mathcal{R}^n$ to the set of outcomes $Y = f(X)$ in the outcome or objective space $\mathcal{R}^p$ (i.e., as defined in (1.1) the outcome set Y is the image of the set of feasible decisions X under the objective function f).

Optimality for an MOP is typically understood in terms of efficiency, and Pareto optimality according to the partial order $\leq$ is assigned to the objective space $\mathcal{R}^p$. Using this partial order, the efficiency of a feasible decision $\hat{x} \in X$ and the Pareto optimality of $\hat{y} = f(\hat{x}) \in Y$ are defined below.

Definition 1.1.9. *Let $X \subseteq \mathcal{R}^n$ be a nonempty feasible set, $f : X \to \mathcal{R}^p$ be an objective* *function, and* $Y = f(X) \subseteq \mathcal{R}^p$ *be the set of outcomes of* X *under* f*. A feasible decision* $\hat{x} \in X$ *is said to be*

(i) efficient *if there does not exist* $x \in X$ *such that* $f(x) \leq f(\hat{x})$*, and*

(ii) weakly efficient *if there does not exist* $x \in X$ *such that* $f(x) < f(\hat{x})$*.*

In these cases, the outcome $\hat{y} = f(\hat{x}) \in Y$ *is said to be a Pareto and a weak Pareto outcome, respectively.*

The set of solutions $\hat{x}$ efficient for problem (1.5) is denoted by $E(X, f, \mathcal{R}^p_{\geqq})$, and the corresponding outcome set, denoted by $N(Y, \mathcal{R}^p_{\geqq})$, is referred to as the Pareto set. Furthermore, the set of weakly efficient solutions $\hat{x}$ for problem (1.5) is denoted by $E_w(X, f, \mathcal{R}^p_{\geqq})$, and the corresponding outcome set, denoted by $N_w(Y, \mathcal{R}^p_{\geqq})$, is referred to as the weak Pareto set. If we have a general cone C, the Pareto set $N(Y, \mathcal{R}^p_{\geqq})$ reduces to the nondominated set $N(Y, C)$ introduced in Definition 2.2.3. In this case, the corresponding efficient set is denoted as $E(X, f, C)$.

The supported efficient solutions and Pareto points of an MOP can be found by solving the single objective optimization problem (SOP) obtained through a linear combination of different objective functions.

Definition 1.1.10. *Let* (X, f) *be an MOP and* $E(X, f, \mathcal{R}^p_{\geqq})$ *be the efficient set and* $N(Y, \mathcal{R}^p_{\geqq})$ *be the Pareto set. Let* $x \in E(X, f, \mathcal{R}^p_{\geqq})$*. If there is some* $\lambda \in \mathcal{R}^p_{\geq}$ *such that* $x \in E(X, f, \mathcal{R}^p_{\geqq})$ *is an optimal solution of min* $\lambda^T f(x)$ *s.t.* $x \in X$*, then* x *is called a* supported efficient solution*, and* $y = f(x)$ *is called a* supported Pareto point*. The sets of all supported efficient solutions and supported Pareto points are denoted by* $E_s(X, f, \mathcal{R}^p_{\geqq})$ *and* $N_s(Y, \mathcal{R}^p_{\geqq})$*, respectively. Otherwise* x *and* y *are called nonsupported points, and their sets are denoted by* $E_{ns}(X, f, \mathcal{R}^p_{\geqq})$ *and* $N_{ns}(Y, \mathcal{R}^p_{\geqq})$*, respectively.*

The efficient set $E(X, f, \mathcal{R}^p_{\geqq}) = E_s(X, f, \mathcal{R}^p_{\geqq}) \cup E_{ns}(X, f, \mathcal{R}^p_{\geqq})$, and the Pareto set $N(Y, \mathcal{R}^p_{\geqq}) = N_s(Y, \mathcal{R}^p_{\geqq}) \cup N_{ns}(Y, \mathcal{R}^p_{\geqq})$.

The ideal point and the nadir point of an MOP are obtained by individually solving the SOPs

$$\begin{aligned} \min\ f_i(x),\ \text{s.t. } x \in X \text{ and} \\ \max\ f_i(x),\ \text{s.t. } x \in X \end{aligned} \tag{1.6}$$

respectively, for $i = 1, \ldots, p$ and combining the optimal objective values. The components of a nadir and an ideal points define the upper and lower bounds for the objective function values of Pareto points, respectively.

Definition 1.1.11. *Let (X, f) be an MOP. The point $y = (y_1, \ldots, y_p) \in \mathcal{R}^p$ with*

$$\begin{aligned} y_i = \min\ f_i(x) \\ s.t.\ x \in X\ for\ \ all\ i = 1, \ldots, p \end{aligned} \tag{1.7}$$

is defined as the ideal point*, and any $r \in \mathcal{R}^p$ with*

$$r \leq f(x)\ for\ \ all\ x \in X$$

is called a utopia point *of the MOP.*

In the following section we discuss methods for computing solutions for MOPs.

1.1.3 Solution approaches for MOPs

The primary goal of multiobjective optimization is to find efficient solutions or nondominated points of an MOP. Thus, it is of interest to design methods for obtaining a complete or partial description of both the efficient and the nondominated sets, referred to in this dissertation as the solution sets. Past research on multiobjective optimization has identified a number of approaches characterizing these two solution sets as well as providing a variety of methods for generating them. These approaches can be categorized into two groups, exact methods and nonexact methods. Exact methods provide the solutions of an MOP by solving it exactly. Nonexact methods provide feasible solutions of an MOP that are not necessarily in the solution sets. The exact methods are classified into two categories, scalarization methods and nonscalarizing methods ([25], [27]).

Scalarization methods convert the vector-valued objective function f of an MOP into a scalar-valued function f, thereby forming a SOP for which the notion of optimality follows from the order of real numbers. Each SOP instance produces one or more solutions for the corresponding MOP. In addition, under some assumptions the optimal solutions of the SOP are efficient solutions for the MOP. Thus, by choosing different scalarization parameters, multiple SOPs are formulated for a single MOP, their optimal solutions corresponding to a subset of the efficient solutions of the MOP.

The most common scalarization method combines all objective functions in the form of a weighted sum ([29]).

Definition 1.1.12. *Let (X, f) be an MOP and $\lambda \in \mathcal{R}^p_{\geq}$. The SOP*

$$\begin{aligned} &min \sum_{i=1}^{p} \lambda_i f_i(x) \\ &s.t.\ x \in X \end{aligned} \tag{1.8}$$

is defined as the weighted-sum scalarization *of the MOP with the weighting parameter λ.*

Based on its geometric interpretation, this method finds the supported Pareto points in the outcome set Y at which the weighting vector λ is normal to a supporting hyperplane to Y ([29]).

A scalarization method originally introduced by Zeleny in 1973 ([111]) utilizes the weighted-l_p norms for $1 \leq p \leq \infty$. The use of the l_∞- norm corresponds to the weighted-Chebyshev method ([12]).

Definition 1.1.13. *Let (X, f) be an MOP and $r \in \mathcal{R}^p$ with $r \leq f(x)$ for all $x \in X$ be a* *utopia point, and* $\lambda \in \mathcal{R}^p_{\geq}$. *The SOP*

$$\begin{aligned} \min \max_{i=1,\ldots,p} \{\lambda_i(f_i(x) - r_i)\} \\ s.t.\ x \in X \end{aligned} \tag{1.9}$$

is referred to as the weighted-Chebyshev scalarization *of the MOP with the reference point* r *and the weighting parameter* λ.

In this dissertation, the weighted-Chebyshev method is used to find all efficient and weakly efficient solutions for an MOP. Several other scalarization methods, are found in the literature, including the ϵ-constraint method ([29]) and Benson's method ([29]) among others.

In contrast to scalarization methods, nonscalarizing methods do not explicitly use a scalarization function but rather use different orders in $\mathcal{R}^p$ to compare objective function values. These orders include the max-order, the lexicographic order, and others. Under some assumptions these methods also provide efficient solutions of an MOP.

The max-ordering problem is a well-known nonscalarizing method and its underlying concept is to minimize the worst objective function value ([29], [57]).

Definition 1.1.14. *Let* (X, f) *be an MOP. The SOP*

$$\begin{aligned} \min \max_{i=1,\ldots,p} \{f_i(x)\} \\ s.t.\ x \in X \end{aligned} \tag{1.10}$$

is referred to as the max-ordering *problem of the MOP.*

It is possible to include a weight vector $\lambda \in \mathcal{R}^p_{\geq}$ in the max-ordering problem (1.10), so that this problem becomes a weighted-max-ordering problem ([29]).

Definition 1.1.15. *Let (X, f) be an MOP and $\lambda \in \mathcal{R}^p_{\geq}$. The SOP*

$$\begin{aligned} \min \max_{i=1,\ldots,p} \{\lambda_i f_i(x)\} \\ s.t.\ x \in X \end{aligned} \tag{1.11}$$

is referred to as the weighted-max-ordering *problem of the MOP.*

The optimal solutions of problems (1.10) and (1.11) are weakly efficient solutions for the corresponding MOP ([29]).

The lexicographic method, another nonscalarizing method, uses the ranking of the objectives in the sense that optimization of the function f_k is only considered once optimality for objectives $\{1, \ldots, k-1\}$ has been established, meaning that objective f_1 has the highest priority and only in the case of multiple optimal solutions with respect to f_1 will objectives f_2 and further objectives be considered. This priority ranking implies the absence of trade-offs between criteria. For example, an improvement in objective f_k does not compensate for the deterioration of any $f_i, i < k$.

In addition to these methods, other nonscalarizing methods can be found in the literature including the lexicographic max-ordering approach ([8]), the equitability approach ([14]), and the balance and level set approaches ([36]) among others.

Many other exact methods have been developed to compute exact solutions for MOPs using the methods discussed above and others found in the literature. Some well researched exact methods are discussed by Ehrgott [29], Jahn [51], Martin et al. [67], Miettinen [69], and among others. The recent exact methods include those developed by Efremov and Kamenev [23], Goel et al. [40], and Hartikainen et al. [44, 45].

Under certain conditions, it is theoretically possible to generate all the solutions of the efficient set and the nondominated set of an MOP using the methods discussed above as well as others found in the literature; however, it is computationally challenging and expensive to obtain these points for various reasons. For an MOP with continuous objective functions and constraints defined over a continuous feasible set, the nondominated set is usually infinite. For an MOP with a discrete feasible set, the nondominated set may have a finite number of elements, but its computation may involve solving an NP-hard combinatorial optimization problem ([89]), the solution set of which is exponential in the size of the test instance of the worst case ([24]).

In conclusion, because of the difficulties faced with obtaining the efficient and the nondominated sets of an MOP using exact methods, the computation of these solution sets needs to be restricted to only a subset of the complete solution set in the form of a discrete representation or a collection of solution points ([6], [97]). The quality of discrete representations of the solution sets is discussed by Faulkenberg and Wiecek [31].

Since the exact solution sets are often not obtainable, various nonexact approaches to characterize or approximate solutions of the efficient and nondominated sets either partially or in their entirety have been proposed in the literature. These nonexact approaches can be categorized into two groups, ϵ-approximation methods and approximation methods.

Even though ϵ-approximation methods yield feasible solutions for an MOP that may or may not be in the solution set, a consistent error bound between the approximation set and the solution set for any instance of the MOP can be obtained using these methods. The concept of ϵ-efficiency, which was originally defined by Kutateladze in 1979 ([58]), relaxes the original efficiency of the solutions in Definition 1.1.9. This concept was later independently proposed by Loridan in 1984 ([62]). In 1986 White ([103]) introduced six alternative definitions of ϵ-efficiency, establishing their corresponding relationships. In 2007, Engau and Wiecek ([28]) investigated ϵ-nondominated points for real vector optimization problems using the concept of translated cones.

The concept of ϵ-efficiency has led to the development of ϵ-approximation. While the fundamental meaning of ϵ-approximation is that the elements in the solution set are approximately dominated by the elements of the approximating set and the approximation quality is described by ϵ, various definitions of ϵ-approximation have been proposed in the literature. For the purposes of this discussion, let S denote a subset of the outcome set Y of an MOP.

Reuter [87] calls a subset S of Y an ϵ-approximation if the set $N(Y,C)+\epsilon$, $\epsilon \in \mathcal{R}^p$ and $\epsilon_1 = \cdots = \epsilon_p$, is dominated by S. An approximation S is referred to as an ϵ-approximation in the sense of Ruhe and Fruhwirth [92] if the set $(1+\epsilon)N(Y,C)$ is dominated by S with $\epsilon \in \mathcal{R}$. An adaptation of ϵ-approximation was introduced by Safer and Orlin in 1995 ([94]) and refined by Papadimitriou and Yannakakis in 2000 ([81]). A subset $S \subset Y$ is defined as an ϵ-approximation if S is a set such that for every point in the nondominated set $N(Y,C)$, the set S contains a point that is at least as good approximately within a factor of $(1+\epsilon)$. Necessary and sufficient conditions for computing an ϵ-approximation for discrete MOPs was further examined, leading to the development of a fast approximation scheme by Safer and Orlin in 1995, while in 2000 Papadimitriou and Yannakakis showed that an ϵ-approximate Pareto set can be constructed in time that is polynomial in the size of the test instance and $1/\epsilon$. In 2010, Legriel et al. ([60]) proposed a method for obtaining an ϵ-approximation of the Pareto set based on the Hausdorff distance between this set and the approximating set. They defined a set of points S in $N(Y,C)$ as an ϵ-approximation of the set $N(Y,C)$ if $\rho(N(Y,C),S) \leq \epsilon$ where $\epsilon \in \mathcal{R}^p$ and $\rho(N(Y,C),S)$ is the Hausdorff distance between the set $N(Y,C)$ and the set S. In addition, in 2011, Laumanns and Zenklusen ([59]) proposed two methods for maintaining a sequence of solution sets that converge to ϵ-approximations of a certain quality.

Motivated by the definition of the ϵ-approximation proposed by Papadimitriou and Yannakakis in 2000 ([81]) and others, ϵ-approximation algorithms for approximating the Pareto sets of many discrete MOPs have been developed. In 1990 Ruhe and Fruhwirth ([92]) reported a method for constructing an ϵ-approximation for the biobjective minimum cost flow problem, and in 2001 Diakonikolas and Yannakakis ([22]) developed a 2-approximation for several biobjective problems including the shortest path problem, the spanning tree problem, the knapsack problem and a scheduling problem. Erlebach et al. [30] and Bazgan et al. [7] independently constructed $(1+\epsilon)$-approximations for the multiobjective knapsack problem. In 2004, Angel et al. ([4]) constructed a 1.5-approximation for the biobjective traveling salesman problem, while in 2005 Angel et al. ([5]) and in 2009 Manthey and Ram ([65]) used different assumptions to propose ϵ-approximations for the multiobjective traveling salesman problem.

When developing ϵ-approximation methods, proving the error bound or the approximation quality ϵ is not easy. In addition, in many situations ϵ-approximation methods are time-consuming. For these reasons approximation methods are often preferred to ϵ-approximation ones. Approximation methods yield feasible solutions for an MOP that may or may not be in the efficient or the nondominated sets but do not provide a consistent error bound between the approximation and the solution set.

Approximation methods can be categorized into two groups: heuristic methods and metaheuristic methods. A heuristic method is an empirical search or optimization method applied to obtain solutions of an MOP not necessarily in the solution sets of the MOP but in the feasible set of the MOP. However, a heuristic method does not guarantee a consistent error bound between the approximated solutions and true solutions for all instances of the MOP as its objective is to quickly produce an acceptable solution for the problem at hand. These methods can be derived from theory or experimental experience ([18], [86] and many others). In addition, often heuristics are problem-specific so that a method which works for one MOP cannot be used to solve an MOP of a different type.

In contrast, metaheuristic methods are powerful techniques generally applicable to a wide range of optimization problems including MOPs. These methods, for example evolutionary or genetic algorithms, are general-purpose algorithms that can be applied to solve almost any optimization problem. These methods also provide approximations of the solution set with points that are not necessarily in this set but that are feasible for the MOP and considered acceptable based on the principle or quality criterion used for the approximation ([17], [21], and many others). Often both heuristic and metaheuristic methods are proposed to approximate solutions of discrete MOPs ([54], [80]).

The solution approaches discussed above are summarized in Figure 1.1.

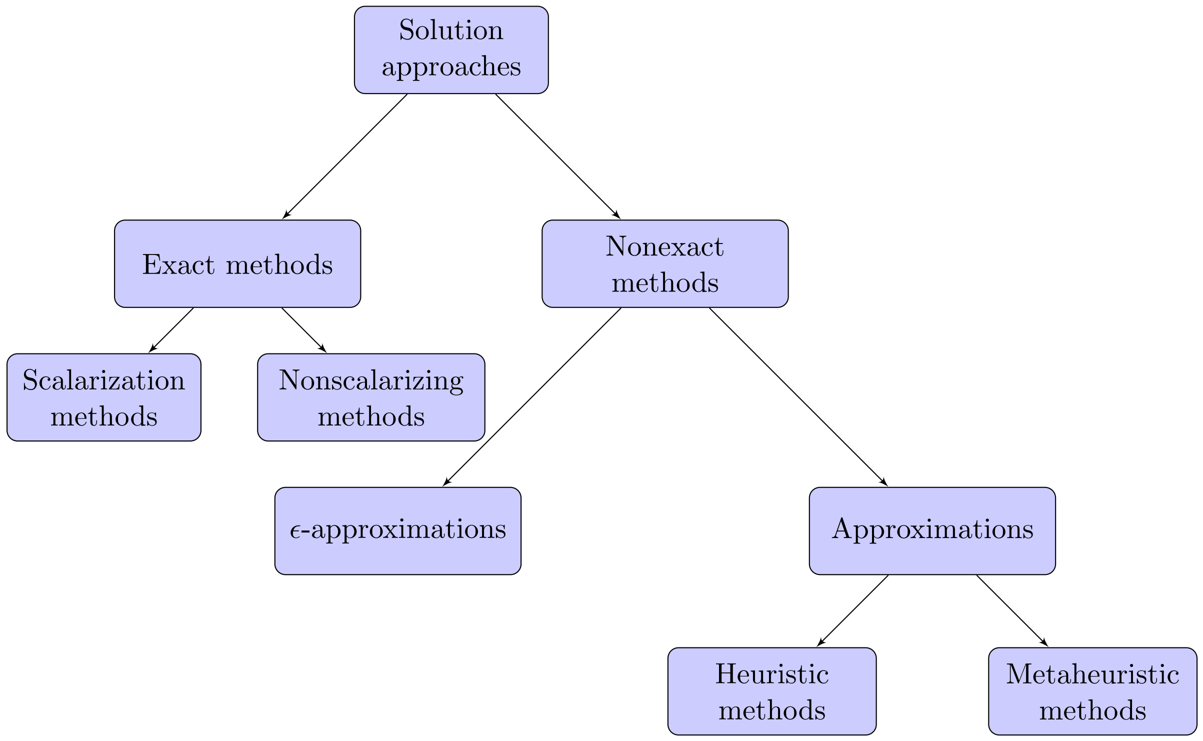


Figure 1.1: Solution Approaches for MOPs

This dissertation further investigates the properties of the ϵ-approximation concept because of the growing interest in it. While the existing theory and the methodology on ϵ-approximation characterize various types of relationships between the approximation and the corresponding solution set, they do not provide a unifying concept for characterizing these relationships applicable to all MOPs. The state of the art in the subject of the ϵ-approximation calls for a study bringing together the proposed concepts and developing an overall theory. In addition, many types of approximation sets have been proposed in the literature, some containing dominated solutions and some not. Thus, grouping or categorizing approximation sets based on the dominated solutions in an approximation set needs to be addressed.

Based on the challenges resulting from obtaining the solution sets, ϵ-approximation, heuristic methods and metaheuristics methods are commonly used to find solutions for multiobjective combinatorial optimization (MOCO) problems, a topic of research interest that has grown recently as evidenced by the articles summarizing those efforts in Ulungu and Teghem ([100]), Ehrgott and Gandibleux ([26]) and Ehrgott ([24]), and others. This dissertation adds to this growing body of research, by investigating and analyzing approximation for the multiobjective set covering problem (MOSCP), one of the challenging MOCO problems that has seen limited study.

1.1.4 The mutiobjective set covering problem

The MOSCP is structured similarly to the well-known single objective set covering problem (SOSCP). An instance of the set covering problem (SCP) consists of a finite set of items and a family of subsets of them such that every item belongs to at least one of the subsets in the family. The goal of the SOSCP is to determine a subset of sets among the sets in the family so that all items are included in at least one set in the subset and the total cost of the selected sets is minimized. When there are p scalar costs for each set in the family, the SCP is called the MOSCP, the formulation of which is given below.

Let E denote the set of items, $E = \{e_1, e_2, \ldots, e_m\}$, with the index set $I = \{i : i = 1, 2, \ldots, m\}$, and S denote a collection of n subsets of E, $S = \{S_1, S_2, \ldots, S_n\}$, with the index set $J = \{j : j = 1, 2, \ldots, n\}$. The items are grouped into subsets of E and the item e_i in E is said to be covered by the set S_j in S provided e_i is in S_j. An instance of the SCP is given by the sets E and S. The binary coefficient a_{ij} for $i \in I$ and $j \in J$ is equal to 1 if the item e_i is covered by the set S_j and is equal to zero otherwise. A cover in this instance is defined as a sub-collection $\{S_j : j \in J^* \subseteq J\}$ which is a subset of S such that all items of E are covered and J^* is the index set of selected sets for the sub-collection. A feasible solution of the SCP requires that each item be covered by at least one selected set. Let $x \in \mathbb{Z}^n$ be the decision variable defined as follows:

$$x_j = \begin{cases} 1 & \text{if } S_j \text{ is selected for a cover} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } j \in J.$$

The set X of all feasible solutions is defined as

$$X = \{x \in \mathbb{Z}^n : \sum_{j \in J} a_{ij} x_j \geq 1 \text{ for } i \in I \text{ and } x_j \in \{0,1\} \text{ for } j \in J\}.$$

The p conflicting objective functions are denoted by $z_q : \mathbb{Z}^n \to R$, with the index set $Q = \{q = 1, \ldots, p\}$. Let $c_j^q > 0$ denote the cost of the set S_j with respect to the objective for $q \in Q$. The goal of the MOSCP is to find a cover such that the costs with respect to all objective functions are minimized. The MOSCP can be represented as follows:

$$\min z(x) = \left[z_1(x) = \sum_{j=1}^{n} c_j^1 x_j, \quad z_2(x) = \sum_{j=1}^{n} c_j^2 x_j, \quad \ldots, \quad z_p(x) = \sum_{j=1}^{n} c_j^p x_j \right] \tag{1.12}$$
$$\text{subject to} \quad x \in X.$$

Since the SCP is categorized as an NP-hard combinatorial problem as shown by Richard in 1972 ([89]) the SOSCP and MOSCP are also NP-hard problems. The SOSCP has been the subject of much study, with various exact and nonexact methods being proposed in literature to solve it ([13], [70]). Chvátal ([15]) and Vazirani ([102]) propose polynomial-time ϵ-approximation algorithms for the SOSCP. Chvátal's ([15]) algorithm has the approximation error ϵ being a function of the cardinality of the largest subset (i.e., $\epsilon = \log m'$ where m' is the cardinality of the largest subset) while Vazirani's ([102]) has it being a function of the number of items in the problem (i.e., $\epsilon = \log m$ where m is the number of items).

The MOSCP, on the other hand, is not a well-studied problem, with only a few nonexact methods (heuristic and metaheuristic methods) found in past research for obtaining its solutions. Liu ([61]) proposes a heuristic algorithm generating only one solution for the MOSCP, a method not of interest here as this dissertation is concerned with obtaining at least a subset of the Pareto set. Saxena and Arora ([96]) formulate the SCP with quadratic objective functions, proposing a method for converting them to linear objective functions by assuming that all objective functions are differentiable and using the Gomory cut technique to arrive at the efficient solutions. Jaszkiewicz ([53], [52]) provides a comparative study of multiobjective metaheuristics for the biobjective SCP (BOSCP), specifically comparing nine well-known multiobjective metaheuristics with a new algorithm, the Pareto memetic algorithm (PMA). This research concluded that the performance of the multiobjective metaheuristics for the BOSCP depends on the problem structure. Prins and Prodhon ([85]) propose a heuristic-based two-phase method (TPM) to find the Pareto set of the BOSCP. In the first phase, the scalarized SCP is solved using a heuristic to generate a subset of the Pareto set referred to as the supported Pareto set. In the second phase, a heuristic algorithm searches for the Pareto points located between two supported Pareto points. This heuristic optimizes one objective function at a time, requiring that the resulting SOSCP be reformulated by Lagrangian relaxation. Lust et al. ([64]) adapt a very large-scale neighborhood search ([3]) for the MOSCP, comparing average running times of the adaptation with the PMA and the TPM for the BOSCP. The performance of their algorithm also depends on the problem structure. General biobjective mixed integer programing methods can be applied as well to solve the BOSCP.

Based on this literature review, the approaches related to obtaining Pareto solutions of the MOSCP involve only heuristic and metaheuristic approaches, none of which can approximate the entire Pareto set, nor is the performance of the algorithms guaranteed. The concept of ϵ-approximation discussed in Section 1.1.3 is used in the approximation of the Pareto set and the formulation of approximation algorithms for many challenging MOCO problems including the traveling salesman problem ([5], [65]), the minimum spanning tree problem ([22]), and the knapsack problem ([7], [30]) but not for the MOSCP. This lack suggests the need for additional research on the MOSCP.

In addition, due to the nature of the heuristic approaches, it cannot be ascertained that one is superior to another, especially since the research on them is limited. Furthermore, these heuristic approaches, when obtaining or improving feasible solutions, do not validate the future costs of the covers associated with these solutions. Thus, developing new heuristic methods to obtain the Pareto set of the MOSCP is merited, especially since the SCP has many real-life applications in such fields such as scheduling, facility location, and designing reserve systems ([20], [75], [88] and many others). In particular, designing reserve systems for ecological species is a common field for its applications as will be discussed in the following section in relation to the MOSCP.

1.1.5 An application of the mutiobjective set covering problem

The goal of the reserve design problem is to find a subset of sites covering a given set of species within a limited conservation budget. Inherently in the majority of such problems, this forms one criterion, while the boundary conditions or structural shapes may lead to additional criteria. In practice, however, reserve design problems need to consider more than just species coverage and budget limitation ([66]). Other spatial characteristics such as the distance between selected reserve sites and the shape of the reserve system should be considered as well. Several mathematical models considering spatial optimization are proposed to address the important issues of the representation of species within reserve systems as seen in [68, 71]. Such approaches make it possible to design a better spatial arrangement for a reserve system by considering attributes such as contiguity and the shape of the selected sites.

Because the SCP formulation does not consider spatial relationships between the sites selected, the resulting reserve system may be highly fragmented, the impact of this fragmentation depending on the specific objectives of the conservation programs. For example, if a reserve system consists of many small habitat areas, it may not facilitate the movement of species among them. As a result, small disconnected reserve systems may be harmful to the survival of the species within the reserve. Moreover, the contiguity of a reserve may be important to species survival within it. Such a reserve system, for example, may help species roam freely within the system without leaving the space.

On the other hand, more compact reserve systems help reduce the edge effects of the system such as the invasion of predators. Also, compact reserve systems help to improve buffering by absorbing disturbances and other adverse impacts. A variety of shape measures have been proposed in reserve selection models to represent compactness, including the boundary length of the reserve, the ratio of boundary length to area, and the average distance between the sites in the reserve system.

As this analysis suggests, contiguity and compactness can be important in modeling reserve site selection problems, and as a result, a variety of formulations have been proposed to address these two attributes. Some explicitly consider both contiguity and compactness, using a linear combination of the two while others consider only one. Shirabe [98] proposes an exact formulation for structural contiguity that can be incorporated into a mixed integer programming model. Based on this model, the resulting system enforces contiguity regardless of other criteria included, such as compactness. Graph theory approaches have also been proposed to control the contiguity of reserve systems. For example, Onal and Briers [78] develop a linear integer programing (IP) formulation using a graph theoretic approach to obtain a connected reserve system. Although this formulation ensures contiguity, it contains what is referred to as gap sites that are to be excluded in the final solution, meaning the objective is to minimize the total number of gap sites. These researchers also incorporate additional variables and constraints explicitly to avoid the formation of cycles. Onal and Wang [79] developed an improved linear IP formulation, also using a graph theory approach, their objective being to minimize the total number of gap sites. In this formulation, a sites are represented by nodes and adjacent sites are represented by arcs. The main difference between these two formulations is the method used to avoid cycle (i.e., a sequence of vertices starting and ending at the same vertex) formation. Although the model in [78] explicitly uses additional constraints and variables to avoid cycles, the improved model [79] does not. Rather, if cycles are present in the solution, new cuts are added, and the model is solved again. The designers of the improved model [79] report that it is computationally more efficient because of its reduced size. While both of these formulations focus on the structural contiguity of a reserve system, Hof and Flather [47] propose a different nonlinear IP model that preserves the contiguity of the system by controlling the shape, requiring reserves to be either circular or rectangular.

In addition, several mathematical models have been proposed to group disconnected sites into compact reserves. In these models [33, 34, 68, 71, 72, 76] reserves of compact shapes are generated as clusters, i.e., collections of adjacent reserve sites. In an ecological sense clusters correspond to different habitats. Separated clusters, i.e., habitats, may be desirable because they will preserve the species in the face of natural disasters such as the destruction of the habitat by fire. In addition, clustering adjacent sites improves the opportunity for multiple biological interactions among a given species.

Onal and Briers [76] develop two integer programming approaches to address the problem of reserve selection to obtain compact reserve systems. In the first, they minimized the sum of the distances between all pairs of sites. In the second approach, an alternative formulation minimizes the largest distance between between selected sites rather than the total distance. More recently, Fischer and Church [33] propose a linear IP formulation for minimizing the boundary length to promote reserve aggregation and compactness.

Fischer and Church [34] propose a bi-objective formulation by considering both the boundary length and the site selection cost while McDonnell et al. [68] models a bi-objective nonlinear IP formulation involving a weighted combination of the boundary length of the selected clusters and the area of the selected sites. They conclude that minimizing the area of the selected sites is equivalent to minimizing their costs. Nalle et al. [71, 72] develop a nonlinear formulation which explicitly addresses the compactness and shape of the selected reserve sites. This model minimizes a weighted combination of two measures: the boundary length of selected clusters and the distance between all pairs of selected sites, even those in disjoint clusters.

The models mentioned above measure the distances between all selected sites whether in the same or different clusters. In some situations consideration of the distance within clusters rather than the distance between all sites may be more useful. For example, if each cluster is being treated as a different habitat, in general it is not important to consider the distance between them. Typically, there are reduced biological interactions among geographically separated clusters. For instance, if one habitat represents a mountain and the other, a swamp, there is no need to measure the distance between these two to obtain compact clusters. Thus, maintaining an optimum distance between all the sites within a given cluster will assure maximum interactions among different sites within it, a factor that has not yet been addressed in the literature. Therefore it is important to minimize the distance within clusters in order to produce compact clusters, a concept that deserves further investigation.

After reviewing several methods for obtaining solutions to MOPs and noting certain difficulties, we conclude that the computation of optimal solutions for an MOP with or without a performance guarantee is a challenging task. Further, we note that obtaining optimal solutions of MOCO problems is even harder due to the numerical complexity of the optimization problem and among the MOCO problems, the MOSCP is not well-studied. The research goals addressing these challenges are given below.

1.2 Research Goals

The research objectives of this dissertation can be classified into three categories: ϵ-approximation for multiobjective optimization problems; approximation of the Pareto set of the multiobjective set covering problem; and an application of the multiobjective set covering problem.

1.2.1 The ϵ-approximation for multiobjective optimization problems

The challenges analyzed in Section 1.1.3 have motivated growing interest in approximating solution sets for MOPs using different notions of ϵ-approximations. To address this interest, this research has the following goals for generalizing, combining and proving the previous and new results of ϵ-approximations to meet the needs of characterizing approximated solution sets:

1. Use the notion of ϵ-approximation to develop a unifying and relevant theory for ap-

proximating the solution sets of an MOP. In particular, cover the variety of results in the literature related to multiple solutions sets, multiple (constant) cones, and multiple quality measures using this unifying concept. In addition, extend the concept of traditional dominance in multiobjective optimization to tolerance-based dominance and identify the properties of approximated solution sets based on this concept.

2. Retrospectively identify the presence of the tolerance-based approximation algorithms found in the literature. In particular, examine the decomposition of complex decision making problems that are modeled as collections of MOPs with respect to their feasible regions and objective functions, and investigate how to use our results obtained in this dissertation for the complex decision making problems.

1.2.2 The multiobjective set covering problem

Because of their nature, the various heuristic approaches for solving MOSCPs discussed in Section 1.1.4 do not provide consistent information about their approximation quality or accuracy, while ϵ-approximation does provide such consistent information. The first objective below addresses the concept of ϵ-approximation for solving MOSCPs, as that has been accomplished previously in such MOCO problems as the traveling salesman problem([4], [65]), the knapsack problem ([7], [30]) and the minimum spanning tree problem ([22]).

1. Develop approximation methods motivated by the concept of ϵ-approximation to obtain the Pareto points of the MOSCP and provide proofs of their correctness. Evaluate the performances of these approximation methods using test problems taken from the literature.

Over the past two decades, heuristic algorithms have been used extensively as optimization tools in solving various MOPs, specifically MOCO problems. The primary reasons for their success are their broad applicability and ease-of-use. Thus, the second objective is to develop and analyze a new heuristic algorithm for solving the MOSCP.

2. Propose a heuristic method for solving the MOSCP using concepts that have not been considered by available heuristic approaches. Evaluate the quality of the approximation generated by the proposed algorithm on test problems using various quality measures. Compare the performance of this method with the performance of the PMA proposed by Jaszkiewicz ([53]) whose work provides the only state-of-the-art results available to us.

1.2.3 An application of the multiobjective set covering problem

A number of optimization models and solution approaches have been proposed to design systems of reserve sites for protecting species and their natural habitats as seen in Section 1.1.5. Further improvements of those models related to the shape of the design need to be considered, though few articles address these factors. The research goals motivated by this need are as follows:

1. Propose an optimization model for obtaining spatially compact ecological reserve systems to protect species. Subsequently, conduct numerical experiments to validate the proposed model in terms of the compactness of the connected groups of reserve sites so that species availability and budget constraints are respected.

1.3 Research Contributions

As a result of investigating the research goals stated in the previous section, this study makes the following contributions to the field of multiobjective optimization and applications.

1.3.1 Contributions to the ϵ-approximation for multiobjective optimization problems

The contributions in relation to goals for the ϵ-approximations for MOPs are as follows:

1. A unifying approximation concept is defined based on the concept of ϵ-approximation for an MOP. The notion of ϵ-approximation serves as a reference to describe tolerance as measured by the function $t : \mathcal{R}^p \to \mathcal{R}^p$ acting on the points in the objective space of the MOP. For a given set $Y \subset \mathcal{R}^p$ and a cone $C \subset \mathcal{R}^p$, the function t is defined as a tolerance function if its image elements are dominated by the elements in Y with respect to the cone C. For $y \in Y$, the largest tolerable deterioration based on the function t is represented by $t(y)$. The specific ϵ-approximations that have been proposed in the literature ([4], [7], [22],[30], [65], and others) for the purpose of approximating solution sets are generalized by the tolerance function. The tolerance functions $t : \mathcal{R}^p \to \mathcal{R}^p$ of the form $t(y) = y + \epsilon$ for $\epsilon \in \mathcal{R}^p$ and $t(y) = (1 + \epsilon)y$ for $\epsilon > 0$ are identified from the literature.

2. Two types of approximation sets of the set being approximated are proposed: *t-covers* and *t-approximations*. Given a set $Y \subset \mathcal{R}^p$, a subset S of the set Y is defined as a t-cover if all elements in Y are covered by at least one element in the S which is at least as good as y up to a tolerance defined by the function t. A t-approximation set is defined as an inherently nondominated t-cover (minimal t-cover) [44] of the set Y. The definition of a t-cover (or a t-approximation) implies that it is of interest to cover the set Y whereas the usual purpose is to cover the nondominated set $N(Y, C)$. However, if the S is a cover for the set $N(Y, C)$, then by our definition $S \subseteq N(Y, C)$. It is theoretically possible to define S as a subset of $N(Y, C)$, but it is often computationally challenging to obtain it, which is reflected in the results obtained for the MOSCP and discussed in Section 1.3.2. It should be mentioned that the definition of a t-cover is different from the definition of a cover defined for the MOSCP in Section 1.1.4. For that problem, the term cover is used to refer to a feasible solution.

3. Properties of t-covers and t-approximations are examined and characterized in a broader context of multiple solutions sets, multiple (constant and polyhedral) cones, and multiple quality measures. Multiple solution sets may result from a decomposi-

tion of the original MOP into smaller problems ([38]); multiple cones may account for different decision makers having different preferences, while multiple quality measures may result from applying different algorithms on the same problem.

4. The traditional concept of dominance given in Definition 1.1.7 is relaxed and extended using a tolerance function, leading to t-dominance. Given a set $Y \subset \mathcal{R}^p$, a cone $C \subset \mathcal{R}^p$ and a tolerance function $t : \mathcal{R}^p \to \mathcal{R}^p$, a point $y^1 \in Y$ is defined as a t-dominated point of the set Y with respect to the function t and the cone C if there exists a point $y^2 \in Y$ such that $y^1 \leqq_C t(y^2)$. The domination set and the t-nondominated set of the set Y based on t-dominance are defined. The relations between the nondominated set and the t-nondominated set of the set Y are identified under multiple quality measures to provide the means of obtaining the t-nondominated set.

5. For the case of Y being a subset of $\mathcal{Z}^p_>$, certain conditions of a tolerance function are proposed to show that under those conditions the t-nondominated set of the set Y is reduced to the Pareto set of the set Y. This result is significant for discrete or combinatorial MOPs when their outcome sets are subsets of $\mathcal{Z}^p_>$.

6. The tolerance functions of the approximation algorithms proposed for approximating the Pareto sets of many MOPs ([4], [7], [22],[30], [65], and others) are identified since these algorithms implicitly make use of certain tolerance functions.

7. The usefulness of the properties of t-covers and t-approximations is demonstrated using complex decision making problems modeled as collections of MOPs ([38], [39]). Using the properties of covers and approximations obtained, constructing an approximation set for the complex system is explained.

1.3.2 Contribution to the multiobjective set covering problem

The contributions to the goals for approximating the solutions in the Pareto set of the MOSCP are as follows:

1. Following the concept of ϵ-approximation, two approximation algorithms are proposed to approximate the supported and weak Pareto points of a MOSCP with a specified accuracy. The approximated solution sets are defined as subsets of the outcome set Y. The accuracy of each algorithm is proven based on the theory used for its development.

2. The first algorithm is adapted from the algorithm proposed for the SOSCP ([102]) to the weighted-sum scalarization of the MOSCP. This algorithm obtains a subset S in Y approximating the supported Pareto set $N_s(Y, \mathcal{R}^p_{\geqq})$ of the MOSCP with a specified accuracy, where Y is the outcome set of the MOSCP. The scalar-valued tolerance function $t : \mathcal{R}^1 \rightarrow \mathcal{R}^1$ is defined as $t(\sum_{i=1}^p y_i) = (1+\epsilon)\sum_{i=1}^p y_i$ for $y \in Y$, $\epsilon > 0$, that is for $y \in Y$, the largest tolerable deterioration based on this function is represented by $(1+\epsilon)\sum_{i=1}^p y_i$ for $y \in Y$. The set S is a $(1+\epsilon)$-approximate supported Pareto set P_s^ϵ. This set, P_s^ϵ, is defined as a subset of Y rather than a subset of $N_s(Y, \mathcal{R}^p_{\geqq})$, which is required by the definition of the t-cover given in Section 1.3.1. Therefore, the approximation set P_s^ϵ is not a t-cover in the sense of the definition given in Section 1.3.1 but is a t-cover in the sense of a relaxed definition. Proofs are given to show that for every supported Pareto point of the MOSCP, there exists a vector $\lambda \in \mathcal{R}^p_{>}$ and a point in the cover S with a specified accuracy defined by ϵ. It is proved that the ϵ depends on the number of items m in the MOSCP.

3. The second algorithm, which is motivated by the first, is developed using the concept of weighted-max-ordering so that the MOSCP is solved in the vector form without scalarization. This algorithm obtains a subset S in Y approximating the weak Pareto set $N_w(Y, \mathcal{R}^p_{\geqq})$ of the MOSCP with a specified accuracy, where Y is the outcome set of the MOSCP. The vector-valued tolerance function $t : \mathcal{R}^p \rightarrow \mathcal{R}^p$ is defined as $t(y) = (1+\epsilon)y$ for $\epsilon > 0$ and $y \in Y$. The set S is a $(1+\epsilon)$-approximate Pareto set P^ϵ. This set, P^ϵ, is defined as a subset of Y rather than a subset of $N_w(Y, \mathcal{R}^p_{\geqq})$, which is required by the definition of the t-cover given in Section 1.3.1. Therefore, the approximation set P^ϵ is not a t-cover in the sense of the definition given in Section

1.3.1 but is a t-cover in the sense of a relaxed definition. Proofs are given to show that for every weak Pareto point of the MOSCP, there exists a vector $\lambda \in \mathcal{R}^p_>$ and a point in the cover S with a specified accuracy defined by ϵ. It is proved that the ϵ depends on the magnitude of the cost coefficients of the MOSCP, the components of λ, and the number of items m in the MOSCP.

4. The proposed algorithms are applied to biobjective SCPs generated by Gandibleux ([1]), and the effectiveness and correctness of each algorithm is verified by the computational results. The experimental performance measures, which are proposed as counterparts to the factors, confirm and improve the theoretical results.

5. A two-phase heuristic algorithm, referred in this dissertation as the Add-Improve Algorithm (AIA), is proposed to approximate the Pareto set of the MOSCP based on two scalarization methods: the weighted-sum method and the weighted-Chebyshev method. The former is used to approximate the supported Pareto points $N_s(Y, \mathcal{R}^p_{\geqq})$ while the latter is used to approximate the nonsupported Pareto points $N_{ns}(Y, \mathcal{R}^p_{\geqq})$, where Y is the outcome set of the MOSCP. In the first phase of the algorithm, a set of initial feasible solutions is found while in the second phase of the algorithm, objective values corresponding to the initial solutions are improved. Unlike for the methods discussed in Section 1.1.4, a merit function is used to estimate the value of the objective functions in the first phase.

6. The AIA is applied to biobjective SCPs generated by Gandibleux ([1]) and to random three-objective SCPs. The test results are used to compare the AIA and the PMA proposed by Jaszkiewicz ([53]). The quality of the approximations generated by the AIA and the PMA is compared using the *Chebyshev-scalarization measure* ($\mathcal{C}$ measure) proposed in [53]. In addition, the *hyper-volume measure* ($\mathcal{H}$ measure) proposed in [112] is used to further evaluate the performance of the AIA. Experimental results confirm that the AIA performs better on a majority of the test problems.

1.3.3 Contributions to applications of the multiobjective set covering problem

The contributions in relation to the goals in Section 1.1.6 are as follows:

1. A biobjective optimization model for selecting reserve sites which clusters them into a relatively small number of compact groups, referred as clusters, is proposed. This model is developed to obtain spatially compact clusters by considering two factors: minimizing both the boundary length and the total distances between all pairs of sites within a reserve system. For the long-term success of a planned reserve system, each of these objectives is important. Because minimizing the boundary length is more important than minimizing the total distance when obtaining compact clusters, the proposed model is formulated as a hierarchical optimization model. The weighted-sum scalarization method is used to appropriately weigh the two hierarchical objectives. The combining weight λ is specified so as to give priority to minimizing the boundary length as the primary criterion.

2. Since the proposed optimization model has non-convex objective functions defined over a discrete feasible region, in order to solve the model more efficiently, the objective functions are linearized. The model is subsequently simplified to reduce the computational effort.

3. To validate correctness of the models, they are applied to randomly generated data sets and to a standard data set based on Oregon field data ([19]). The experimental results show that the proposed simplifications significantly improve the computational effort. In addition, computational results confirm that models provide compact clusters which help to protect species and their natural habitats.

1.4 The content of the dissertation

For further guidance, we now give a brief outline of the general organization of the dissertation.

Chapter 2, which presents the contributions outlined in Section 1.3.1, begins with the current theories about ϵ-approximation and the new unifying concepts of a tolerance function t, a t-cover and a t-approximation set. It then goes on to investigate the properties of t-covers with respect to polyhedral cones, general cones, multiple solution sets, and multiple tolerance functions. Section 2.4 includes the properties relevant for t-approximation sets. Identifying tolerance-based approximation algorithms in the literature and investigating how to use the results obtained for complex decision making problems are discussed in Section 2.5. The concept of tolerance-based dominance, referred to as t-dominance, and the new properties identified using it are discussed in Section 2.6.

Presenting the contributions related to approximating the Pareto set of the MOSCP using ϵ-approximation, Chapter 3 begins with the theoretical approaches used to find the efficient solutions of the MOSCP and the ϵ-approximation concept. Section 3.3 develops two algorithms for approximating this set: the first aims to approximate the supported Pareto set and the second, to approximate the weak Pareto set. In particular, the accuracy of the solutions produced by both algorithms are proven in Section 3.3, the computational results being included in Section 3.4.

Chapter 4 provides a two-phase heuristic algorithm for approximating the Pareto set of MOSCPs. More specifically, Section 4.3 presents the fundamental ideas and observations used in the development of the algorithm. The proposed algorithm is explained in Section 4.4, and its performance and its comparison with the PMA proposed by Jaszkiewicz ([53]) are discussed in Section 4.5.

The paper presented in Chapter 5 explores the application of the SCP discussed in Section 1.1.5. As indicated by the contributions suggested in Section 1.3.3, the formulation of the mathematical model for obtaining compact reserve sites is given in Section 5.2. The second model proposed for solving the initial model more efficiently is also presented in this section. Numerical examples comparing the computational aspects of the two models are provided in Section 5.3, and additional data sets including a standard one based on the Oregon field data ([19]) are then used in Sections 5.4 to explore more fully the computational behavior of the models as their parameters are varied.

1.5 Conclusion and future research

Multiobjective optimization is an area of multiple criteria decision making concerned with mathematical problems involving more than one conflicting objective functions needing to be optimized simultaneously. Because of these conflicting objective functions, the solution involves a set rather than a single solution for these problems. Obtaining this set is difficult for many reasons, and, therefore, many approaches for approximating them have been proposed. The research presented in this dissertation addresses these observations and difficulties, its primary contribution focusing on two areas: the first is the solution approximation and theoretical characterization of the solution sets of an MOP and the second focuses on the approximation of the Pareto set of the well-known MOSCP, which has received limited study, and its application in the field of reserve design.

Chapter 2 presents a unified approach for representing ϵ-approximations using the tolerance function t and two types of approximation sets, t-covers and t-approximations, and some of their properties. Future work could explore additional properties, including applying them to a specific MOP for which there is an ϵ-approximation algorithm. The relationships between the nondominated set and the t-nondominated set of an MOP are analyzed using multiple quality measures. These results enable and stimulate further investigation of approximate solutions for the MOP solution, approaches proposed in this dissertation impacting their future use and study.

In Chapter 3 two algorithms for approximating the Pareto set of the MOSCP are developed and their approximation quality derived. Algorithm 1 approximates the supported

Pareto points with a constant error depending only on the number of items in the test instance. It is shown that for every supported Pareto point, there exists a vector $\lambda \in \mathcal{R}^p$ and a point in the approximated solution set. Algorithm 2 approximates the weak Pareto points with a known quality depending on the problem data (number of items and the magnitude of the cost coefficients) and on the weight vector used for computing the approximation, the results indicating that for every weak Pareto point, there exists a vector $\lambda \in \mathcal{R}^p$ and a point in the approximated solution. Even though the results show that such λ vectors exist, they do not reveal how this vector can be found. Thus, an important avenue of future study is how these λ vectors can be obtained. Algorithm 2 is the first in the literature to approximate the Pareto points of the MOSCP with known approximation quality. Even though, this algorithm provides approximation points with known quality ϵ, these points are far from the Pareto points as seen in Section 3.4. Therefore, another possible direction for future work includes approximation concepts other than the one chosen for Algorithm 2.

Chapter 4 proposes a two-phase heuristic algorithm, the AIA, for approximating the Pareto points of the MOSCP, developed based on the weighted-sum and the weighted-Chebyshev methods. In the first phase of the AIA, a set of initial feasible solutions is found while in the second phase, objective values corresponding to initial solutions are improved. The performance of this proposed AlA is compared with the performance of the PMA developed by Jaszkiewicz ([53]), the results indicating that the AlA performs better in many test instances. However, as discussed in Section 1.1.3, due to their nature, it cannot be proved that one heuristic algorithm is superior to another. Thus, there are several directions for future research on this topic. To enhance the quality of the initial feasible solutions, it is possible to consider optimization methods other than the one considered in the first phase of the proposed algorithm. In addition, sophisticated improvement strategies could be added to obtain improved solutions in the second phase. Moreover, it would be useful to conduct additional and more extensive testing of this algorithm on large problems and problems with more than three objective functions.

Chapter 5 investigates the design of spatially compact reserve systems, addressing the limitations of the standard SCP covering approach. A hierarchical optimization model that organizes reserve sites into a relatively small number of compact groups, or clusters, is developed, one that explicitly considers two factors: minimizing the boundary length of all selected clusters and minimizing the total distance between all pairs of sites within each cluster. Each of these objectives is important for the long-term success of a planned reserve system. Because the boundary length is more important than the total within cluster distance when creating desirable clusters, the hierarchical optimization model gives priority to minimizing the boundary length. Because the initial formulation is expressed as a non-convex optimization model defined over a discrete feasible region, the model is linearized in order to solve it more efficiently. The experimental results show that the linearized model significantly reduces the computational time. Both models provide compact clusters to better protect species. Because it may be difficult to find an optimal solution for the linearized model in a reasonable amount of time for large reserve design problems, future work could involve developing appropriate heuristic algorithms (e.g., see [16, 55]).

This dissertation extends the investigation of multiobjective optimization, especially multiobjective combinatorial optimization. It is hoped that the questions explored in this dissertation and the subsequent results and their implications add to the understanding of the field as well as open new areas of focus for futer research.

Chapter 2

Covers and t-dominance in Multiobjective Optimization

2.1 Introduction

Multiobjective optimization problems (MOPs), which occur frequently in many real-world applications, involve optimizing several objective functions over a feasible region defined by constraint functions. Because the objective functions are often conflicting, a unique solution that optimizes all of them simultaneously does not exist. Instead, an optimal solution set, also known as the nondominated set, is implied by a partial order or a convex cone associated with the objective space of the MOP ([110]). Traditionally, the partial order is based on the Pareto preference ([82]) which, in case of minimization, is equivalent to the cone being the first orthant of the objective space ([27]). It has been shown in the literature that general convex cones are also beneficial since they provide a tool for modeling decision maker's preferences. Refer to Noghin ([73]), Noghin and Tolstykh ([74]), Hunt and Wiecek ([50]), Klimova and Noghin ([56]), Wiecek ([104]), and Hunt et al. ([49]) for algebraic models of such polyhedral cones, and to Hunt et al. ([48]) and Blouin et al. ([10]) for their applications in engineering design.

The success of applying multiobjective optimization in practice depends, among other things, on the ability to compute the elements of the nondominated set. This set is typically large and it is often difficult or even impossible to obtain its exact description. For MOPs with continuous objective and constraint functions defined over a continuous feasible set, the nondominated set is usually infinite. It can be derived analytically only for certain classes of problems ([99, 41]), and otherwise it has to be approximated. For MOPs with a discrete feasible set, the nondominated set may have a finite number of elements. However, their computation may involve solving *NP*-hard problems. In particular, for most multiobjective combinatorial optimization problems the solution set is exponential in the size of a test instance in the worst case ([29]). In conclusion, even if it is theoretically possible to find the nondominated set, it is often computationally challenging and expensive to do so.

Because of the difficulties related to the structure of the nondominated set and the compounding computational challenges, a wide variety of methods have been proposed for computing its elements, and numerous approaches have been developed to approximating this set. All these methods and approaches rely on exact or heuristic algorithms. The former typically compute actual elements of the solution sets by means of algorithms which provide theoretical proofs for their correctness (refer to ([93]) for a review of such methods for continuous MOPs in the period 1975–2003). Recent exact methods include the works of [67, 40, 23], and [44, 45]. The quality of discrete representations is discussed in [31]. Heuristic and meta-heuristic methods (such as evolutionary and genetic algorithms) provide approximations of the nondominated set with points that are not necessarily in this set but that are feasible for the MOP and considered acceptable according to a principle or a quality criterion used for the approximation. These methods usually find approximating points quickly but have no proofs of correctness and, thus, are theoretically unsupported (see [21, 17], and many others).

Rules or criteria for evaluating approximation quality evolved from the concept of ϵ-nondominance that was introduced by Kutateladze ([58]) to relax the original efficiency of the solutions. Other concepts of ϵ-nondominance followed. They had been researched as types of efficiency (refer to [46] for a survey) before they were used for approximation. The general idea behind ϵ-approximations is that the elements in the solution set are approximately dominated by the elements of the approximating set and the approximation quality is a function of ϵ.

Discrete ϵ-approximations of the solution set of the MOP have been proposed by various authors. Motivation for those approaches and an overview focusing on defining and measuring approximation quality is discussed by Sayin ([97]). The problem of computing an ϵ-approximation for discrete MOPs is examined by Safer and Orlin ([94]) and Papadimitriou and Yannakakis ([81]). The former develop necessary and sufficient conditions for the existence of a fast approximation scheme while the latter show that an ϵ-approximate Pareto set can be constructed in time that is polynomial in the size of a test instance and $1/\epsilon$. Often, problem-based ϵ-approximations are constructed in the literature. Diakonikolas and Yannakakis ([22]) develop a 2-approximation for several biobjective problems such as the shortest path problem, the spanning tree problem, the knapsack problem, and a scheduling problem. Erlebach et al. ([30]) and Bazgan et al. ([7]) independently construct $(1+\epsilon)$-approximations for the multiobjective knapsack problem. Angel et al. ([4]) construct a 1.5-approximation for the biobjective traveling salesman problem, while Angel et al. ([5]) and Manthey and Ram ([65]) use different assumptions to propose ϵ-approximations for the multiobjective traveling salesman problem. Other authors address general MOPs. Existence of approximations under different assumptions is discussed by Vassilvitskii and Yannakakis ([101]). Legriel et al. ([60]) propose a method for obtaining ϵ-approximation based on the Hausdorff distance between the Pareto set and the approximating set. Laumanns and Zenklusen ([59]) present two methods for maintaining a sequence of solution sets that converge to ϵ-approximations of a certain quality. Filippi and Stevanato ([32]) describe two $(1+\epsilon)$-approximation methods for approximating the Pareto set of biobjective combinatorial optimization problems and test the proposed methods on the biobjective traveling salesman problem.

Due to the growing interest in approximation for MOPs and many different notions of ϵ-approximations in the literature, a theoretical framework for defining and classifying sets representing or approximating solution sets for MOPs is proposed in this paper. The notions of ϵ-efficiency serve as a reference to construct a function acting on the elements in the objective space of the MOP. This function, which is used to compare the elements being approximated with the elements that are approximating, models an associated quality, or more generally, a tolerance measure. Two types of subsets of the set being represented or approximated are defined: *covers* and *approximations*. A cover is defined as a subset of a set of interest so that the elements of the cover remain in a relation with the elements of the set. This relation is defined for every element in the set of interest using a tolerance function that also determines the error or quality of the cover. An approximation set is defined as an inherently nondominated cover ([44]).

Covers and approximations are studied in a broader context of multiple solutions sets, multiple (constant) cones, and multiple quality measures. Multiple solution sets may result from a decomposition of the original MOP into smaller problems ([38]); multiple cones may account for different decision makers having different preferences; while multiple quality measures may result from using different algorithms on the same problem. Properties of covers and approximations are derived for a variety of these cases representing different real-life circumstances. While the notions of ϵ-dominance had led researchers to the development of various types of approximations, the proposed tolerance function extends the traditional dominance ([110]) to t-dominance in which the dominating element is replaced by its proxy or surrogate that is yield by the tolerance function.

The chapter is organized in the following manner. In Section 2.2, common terminology and basic definitions are given, and the definitions of cover and approximation are presented. Their properties are studied in Sections 2.3 and 2.4. Relevance of the results to the approximation algorithms in the literature and an application are given in Section 2.5. In Section 2.6, the notion of t-dominance is investigated.

2.2 Notations and definitions

We begin with well established notations and definitions. Throughout this paper let $\mathcal{R}^p$ be a Euclidean vector space, Y be a nonempty subset in $\mathcal{R}^p$, and C be a nonempty cone in $\mathcal{R}^p$. A set C in $\mathcal{R}^p$ is called a cone if $d \in C$ then $\lambda d \in C$ for $\lambda \geq 0$.

For y^1, y^2 in Y, we use the notation $y^1 \leqq y^2$ if and only if $y^1_k \leq y^2_k$ for all $k = 1, 2, \ldots, p$; $y^1 \leq y^2$ if and only if $y^1_k \leq y^2_k$ for all $k = 1, 2, \ldots, p$ and $y^1 \neq y^2$; $y^1 < y^2$ if and only if $y^1_k < y^2_k$ for all $k = 1, 2, \ldots, p$. With the relations $\geqq, \geq$ and $>$ defined accordingly, we also define the cone $\mathcal{R}^p_{\geqq} := \{y \in \mathcal{R}^p : y \geqq 0\}$.

Cones are used to define cone relations between the elements of the set Y, and dominated and nondominated elements in Y ([110]).

Definition 2.2.1. *Let $y^1, y^2 \in Y$. A relation $\leqq_C$ on Y is defined by $y^1 \leqq_C y^2$ if and only if $y^2 - y^1 \in C$, or equivalently, there exists $d \in C$ such that $d = y^2 - y^1 \in C$. Furthermore a relation $\leq$ on Y is defined by $y^1 \leq_C y^2$ if and only if $y^2 - y^1 \in C \setminus \{0\}$, or equivalently, there exists $d \in C$, $d \neq 0$ such that $d = y^2 - y^1 \in C$.*

Definition 2.2.2. *A point $y' \in Y$ is called a dominated point of the set Y with respect to the cone C if there exists a point $y \in Y$ such that $y \leq_C y'$.*

Definition 2.2.3. *A point $y' \in Y$ is called a nondominated point of the set Y with respect to the cone C if there does not exist $y \in Y$ and $d \in C$, $d \neq 0$ such that $y' = y + d$ or, equivalently, there does not exist $y \in Y$ such that $y \leq_C y'$. The set of all nondominated points of Y with respect to the cone C is denoted by $N(Y, C)$.*

If the cone C is the Pareto cone ($C = \mathcal{R}^p_{\geqq}$), the set $N(Y, C)$ reduces to the well-known Pareto set and is denoted as $N(Y, \mathcal{R}^p_{\geqq})$.

A polyhedral cone is defined as follows:

Definition 2.2.4. *A polyhedral cone is a cone C in $\mathcal{R}^p$ for which there exists a $q \times p$ matrix $A \in \mathcal{R}^{q \times p}$ such that $C = \{d \in \mathcal{R}^p : Ad \geqq 0\}$.*

The image of Y under the linear mapping represented by the matrix $A \in \mathcal{R}^{q\times p}$ is denoted by $A[Y] := \{z \in \mathcal{R}^q : z = Ay \text{ for some } y \in Y\}$.

We now present the concepts newly introduced in this chapter. We begin with a tolerance function that yields proxy or surrogate elements of the elements in the set Y. A function provides tolerance if its image elements, i.e., the proxy elements, are dominated by the elements in Y with respect to a cone C.

Definition 2.2.5. *Let C be a cone in $\mathcal{R}^p$. A vector-valued function $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ such that $y \leqq_C t(y)$ for all $y \in \mathcal{R}^p$ is called a tolerance function.*

Definition 2.2.5 implies that $t(y) \in C + y$ for all y in Y. Otherwise the tolerance function is not well defined.

Example 2.2.1. *For $C = \mathcal{R}^p_{\geqq}$, the function $t : \mathcal{R}^p \mapsto \mathcal{R}^p$, $t = 2y$, is well defined. However, for $C = -\mathcal{R}^p_{\geqq}$, it is not.*

The following property of the tolerance function will be useful.

Definition 2.2.6. *Let Y be a set in $\mathcal{R}^p$. A vector-valued function $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ is called an A-invariant function on Y if $t(Ay) = At(y)$ for all $y \in Y$, where A is a $p \times p$ matrix.*

Example 2.2.2. *Let $Y \subset \mathcal{R}^p$ and $t_1, t_2 : \mathcal{R}^p \mapsto \mathcal{R}^p$ be two functions defined as $t_1(y) = (1+\epsilon)y$ for $\epsilon \in \mathcal{R}_{\geq}$ and $t_2(y) = y + \epsilon$ for $\epsilon \in \mathcal{R}^p_{\geq}$, and A be an $p \times p$ matrix. Then t_1 is an A-invariant function on Y but t_2 is not.*

Example 2.2.3. *Let $Y \subset \mathcal{R}^2$ such that $Y = \{(2,3)^T, (1,1)^T\}$, $t_1, t_2 : \mathcal{R}^2 \mapsto \mathcal{R}^2$ be two functions defined as $t_1(y) = 2y$ and $t_2(y) = 2y + (1,1)^T$ for $y \in Y$, and $A = \begin{pmatrix} 2 & 3 \\ 1 & 2 \end{pmatrix}$. We obtain that $t_1(Ay) = At_1(y)$ for all $y \in Y$. Thus t_1 is an A-invariant function on Y. Let $\bar{y} = (2,3)^T$. We note that $t_2(A\bar{y}) = (27,17)^T$, $At_2(\bar{y}) = (31,19)^T$, and hence t_2 is not an A-invariant function on Y.*

Given $Y \subset \mathcal{R}^p$, a subset S of Y is called a cover of Y if every element in Y is "covered" by an element in S. An element in Y is "covered" if its proxy is dominated by an element in S with respect to a cone C. In other words, the tolerance function implies the level of tolerance according to which the proxy elements are treated as the actual elements of Y even that the proxy elements may not be in Y.

Definition 2.2.7. *Let Y be a set in $\mathcal{R}^p$, C be a cone in $\mathcal{R}^p$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. A subset S of Y is called a t-cover of Y with respect to C if for all $y \in Y$ there exists $s \in S, s \leqq_C t(y)$.*

Example 2.2.4. *Let $Y \subset \mathcal{R}^2$ such that $Y = \{y^1 = (1,3)^T, y^2 = (2,2)^T, y^3 = (3,3)^T, y^4 = (4,1)^T, y^5 = (8,1)^T, y^6 = (9,3)^T\}$, and $t : \mathcal{R}^2 \mapsto \mathcal{R}^2$ be a tolerance function such that $t(y) = 2y$ for $y \in Y$.*

1. *Let $C = \{d \in \mathcal{R}^2 : d_2 \leq 3d_1,\ d_2 \geq \frac{1}{8}d_1,\ d_1, d_2 \geq 0\}$. By Definition 2.2.5, $y^i \leqq_C t(y^i), i = 1, \ldots, 6$. Also note that $y^2 \leqq_C t(y^3) = (6,6)^T$ and $y^4 \leqq_C t(y^6) = (18,6)^T$. Therefore the points y^3 and y^6 are covered by y^2 and y^4, respectively. Thus, the set $S = \{y^1, y^2, y^4, y^5\}$ is a t-cover for Y with respect to the cone C. Further, note that $S \setminus \{y^4\}$ does not form a cover for Y because there is no $s \in S \setminus \{y^4\}$ such that $s \leqq_C t(y^4)$.*

2. *Let $C = \mathcal{R}^2_{\geqq}$ and note that the sets S and $S \setminus \{y^4\}$ are both t-covers for Y with respect to the cone $\mathcal{R}^2_{\geqq}$.*

For the given cone and tolerance function, there may exist many t-covers. The collection of all t-covers of Y with respect to a cone C is denoted by $\mathcal{C}_t(Y, C)$. Some covers in the collection may contain a large number of points while others may be small. Also, the points of a cover may dominate each other or not. By removing all dominated points from a cover, the set remains a cover. If a set S is a t-cover of Y and does not contain any dominated points then this set is referred to as a t-approximation set of Y. In [44], a set S satisfying $N(S, C) = S$ is called an inherently nondominated set.

Definition 2.2.8. *Let Y be a set in $\mathcal{R}^p$, C be a cone in $\mathcal{R}^p$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. A t-cover $S \in \mathcal{C}_t(Y, C)$ such that $N(S, C) = S$ is called a t-approximation set of Y with respect to C.*

Example 2.2.5. *Let Y and t be defined as in Example 2.2.4.*

1. *Let the cone C be defined is in part 1. Thus the set S is a t-approximation sets for Y with respect to C.*

2. *Let the cone C be defined as in part 2. Note that $N(S, C) \neq S$. Thus the set S is not a t-approximation set for Y with respect to C.*

Given the cone and tolerance function, a t-approximation set in general is not unique. The set of all t-approximation sets of Y with respect to a cone C is denoted by $\mathcal{A}_t(Y, C)$.

2.3 Properties of covers

In this section properties of covers are studied with respect to polyhedral and general cones, multiple solution sets, and multiple tolerance functions.

Because the relationship between the nondominated set with respect to a general polyhedral cone and the Pareto cone is well established ([95]), we examine covers with respect to polyhedral cones, sets, and related linear transformations. The first proposition shows the behavior of covers under the linear transformation represented by the matrix of the cone.

Proposition 2.3.1. *Let Y be a set in $\mathcal{R}^p$ and C be a polyhedral cone in $\mathcal{R}^p$, $C = \{d \in \mathcal{R}^p : Ad \geqq 0\}$ where A is a $p \times p$ matrix. Let $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be an A-invariant tolerance function. Then $S \in \mathcal{C}_t(Y, C)$ if and only if $A[S] \in \mathcal{C}_t(A[Y], \mathcal{R}^p_{\geqq})$.*

Proof. ($\Rightarrow$) Let $S \in \mathcal{C}_t(Y, C)$ and $y \in Y$. Then by Definition 2.2.7, there exists $s \in S$ such that $s \leqq_C t(y)$. By Definition 2.2.1, there exists $d \in C$ such that $d = t(y) - s$. Since $d \in C$, $Ad \geqq 0$ and

$$Ad = A(t(y) - s) \geqq 0. \tag{2.1}$$

By setting $d_1 = Ad$, we get $d_1 \in \mathcal{R}^p_{\geqq}$. Then (2.1) can be written as $d_1 = At(y) - As \in \mathcal{R}^p_{\geqq}$, and, by Definition 2.2.1, we have $As \leqq_{\mathcal{R}^p_{\geqq}} At(y)$. Because t is an A-invariant function, we obtain

$$As \leqq_{\mathcal{R}^p_{\geqq}} t(Ay). \tag{2.2}$$

Thus for all $y \in Y$ there exists $s \in S$ such that inequality (2.2) holds. Also, for all $Ay \in A[Y]$ there exists $As \in A[S]$ such that $As \leqq_{\mathcal{R}^p_{\geqq}} t(Ay)$. Thus $A[S] \in \mathcal{C}_t(A[Y], \mathcal{R}^p_{\geqq})$.

($\Leftarrow$) Let $A[S] \in \mathcal{C}_t(A[Y], \mathcal{R}^p_{\geqq})$ for $S \subseteq Y$ and $u \in A[Y]$ with $u = Ay$ for $y \in Y$. Then by Definition 2.2.7 there exists $\bar{u} \in A[S]$ with $\bar{u} = As$ for some $s \in S$, such that $As \leqq_{\mathcal{R}^p_{\geqq}} t(Ay)$, or equivalently $t(Ay) - As \in \mathcal{R}^p_{\geqq}$. Because t is an A-invariant function, we obtain $At(y) - As \in \mathcal{R}^p_{\geqq}$. Since $A(t(y) - s) \geqq 0$, we have $t(y) - s \in C$. Therefore, for all $y \in Y$ there exists $s \in S$ such that $s \leqq_C t(y)$ which completes the proof. □

Corollary 1 addresses the case of Proposition 2.3.1 when the matrix A is replaced by a matrix B not related to the cone C. The proof is analogous to the 'only if' part of the proof of Proposition 2.3.1.

Corollary 1. *Let $Y \subset \mathcal{R}^p$ and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a B-invariant tolerance function where B is a $p \times p$ nonnegative matrix. If $S \in \mathcal{C}_t(Y, \mathcal{R}^p_{\geqq})$ then $B[S] \in \mathcal{C}_t(B[Y], \mathcal{R}^p_{\geqq})$.*

Proposition 2.3.2 extends Proposition 2.3.1 replacing the cone C with a polyhedral set C, $C = C(A, b) = \{d \in \mathcal{R}^p : \ Ad \geqq b\}$, where A is a $p \times p$ matrix and $b \in \mathcal{R}^p$.

Proposition 2.3.2. *Let Y be a set in $\mathcal{R}^p$ and C be a polyhedral set in $\mathcal{R}^p$, $C = C(A, b) = \{d \in \mathcal{R}^p : \ Ad \geqq b\}$, where A is a $p \times p$ matrix and $b \in \mathcal{R}^p$. Let $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be an A-invariant tolerance function. Then $S \in \mathcal{C}_t(Y, C)$ if and only if $A[S] \in \mathcal{C}_t(A[Y], \mathcal{R}^p_{\geqq b})$, where $\mathcal{R}^p_{\geqq b} := \mathcal{R}^p_{\geqq} + b = \{d \in \mathcal{R}^p : d \geqq b\}$.*

Proof. ($\Rightarrow$) Let $S \in \mathcal{C}_t(Y, C)$ and $y \in Y$. Then by Definition 2.2.7, there exists $s \in S$ such that $s \leqq_C t(y)$. By Definition 2.2.1, there exists $d \in C$ such that $d = t(y) - s$. Since $d \in C$, $Ad \geqq b$ and $Ad = A(t(y) - s) \geqq b$. Thus we obtain $Ad = At(y) - As \in \mathcal{R}^p_{\geqq b}$, and again by Definition 2.2.1, we have $As \leqq_{\mathcal{R}^p_{\geqq b}} At(y)$. Since t is an A-invariant function, $As \leqq_{\mathcal{R}^p_{\geqq b}} t(Ay)$. Since y is arbitrary, for all $y \in Y$ there exists $s \in S$ such that $As \leqq_{\mathcal{R}^p_{\geqq b}} t(Ay)$. Also, for all

$Ay \in A[Y]$ there exists $As \in A[S]$ such that $As \leqq_{\mathcal{R}^p_{\geqq b}} t(Ay)$. Thus $A[S] \in \mathcal{C}_t(A[Y], \mathcal{R}^p_{\geqq b})$.

($\Leftarrow$) Let $A[S] \in \mathcal{C}_t(A[Y], \mathcal{R}^p_{\geqq b})$ for $S \subseteq Y$ and let $u \in A[Y]$ with $u = Ay$ for $y \in Y$. Then by Definition 2.2.7 there exists $\bar{u} \in A[S]$ with $\bar{u} = As$ for some $s \in S$, such that $As \leqq_{\mathcal{R}^p_{\geqq b}} t(Ay)$, or equivalently $t(Ay) - As \in \mathcal{R}^p_{\geqq b}$. Since t is an A-invariant function, we obtain $At(y) - As \in \mathcal{R}^p_{\geqq b}$. Since $A(t(y) - s) \geqq b$, we have $t(y) - s \in C$. Therefore, for all $y \in Y$ there exists $s \in S$ such that $s \leqq_C t(y)$, which completes the proof. $\square$

Cones containing the Pareto cone model relative importance of criteria ([73, 49]) and also reduce the nondominated set ([95]), which facilitates the resulting decision making process ([48, 10]). Proposition 2.3.3 shows a relationship between a t-cover of the Pareto set and a t-cover of the nondominated set defined with respect to a polyhedral cone C subsuming the Pareto cone.

Proposition 2.3.3. *Let Y be a set in $\mathcal{R}^p$ and C be a pointed polyhedral cone in $\mathcal{R}^p$, $C = \{d \in \mathcal{R}^p : \ Ad \geqq 0\}$ where A is a $p \times p$ matrix such that $\mathcal{R}^p_{\geqq} \subseteq C$. Let $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S \in \mathcal{C}_t(N(A[Y], \mathcal{R}^p_{\geqq}), \mathcal{R}^p_{\geqq})$ then $S \in \mathcal{C}_t(A[N(Y, C)], C)$.*

Proof. Let $S \in \mathcal{C}_t(N(A[Y], \mathcal{R}^p_{\geqq}), \mathcal{R}^p_{\geqq})$ with $S = A[\bar{S}]$ for some $\bar{S} \subseteq Y$. Since C is a pointed polyhedral cone, $N(A[Y], \mathcal{R}^p_{\geqq}) = A[N(Y, C)]$, and we obtain $S \subseteq A[N(Y, C)]$. Let $u \in A[N(Y, C)]$ with $u = Ay$ for some $y \in Y$. By Definition 2.2.7, there exists $s \in S$ with $s = A\bar{s}$ for some $\bar{s} \in Y$ such that $s \leqq_{\mathcal{R}^p_{\geqq}} t(u)$ or, equivalently, $A\bar{s} \leqq_{\mathcal{R}^p_{\geqq}} t(Ay)$. Then by Definition 2.2.1, we obtain $t(Ay) - A\bar{s} \in \mathcal{R}^p_{\geqq}$. Since $\mathcal{R}^p_{\geqq} \subseteq C$, we obtain $t(Ay) - A\bar{s} \in C$ and hence, again by Definition 2.2.1, we have that $A\bar{s} \leqq_C t(Ay) = t(u)$. Since u is arbitrary, for all $u \in A[N(Y, C)]$ there exists $s \in S$ such that $s \leqq_C t(u)$. Thus the proof is complete. $\square$

Proposition 2.3.4. *Let Y be a set in $\mathcal{R}^p$ and C be a polyhedral cone in $\mathcal{R}^p$, $C = \{d \in \mathcal{R}^p : \ Ad \geqq 0\}$ where A is a $p \times p$ matrix such that $C \subseteq \mathcal{R}^p_{\geqq}$. Let $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S \in \mathcal{C}_t(A[N(Y, C)], C)$ then $S \in \mathcal{C}_t(N(A[Y], \mathcal{R}^p_{\geqq}), \mathcal{R}^p_{\geqq})$.*

Proof. Let $S \in \mathcal{C}_t(A[N(Y, C)], C)$ with $S = A[\bar{S}]$ for some $\bar{S} \subseteq N(Y, C)$. Since C is a polyhedral cone, $A[N(Y, C)] \subseteq N(A[Y], \mathcal{R}^p_{\geqq})$ and we obtain $S \subseteq N(A[Y], \mathcal{R}^p_{\geqq})$. Let

$u \in N(A[Y], \mathcal{R}^p_{\geqq})$ with $u = Ay$ for some $y \in Y$. By Definition 2.2.7, there exists $s \in S$ with $s = A\bar{s}$ for some $\bar{s} \in Y$ such that $s \leqq_C t(u)$ or, equivalently, $A\bar{s} \leqq_C t(Ay)$. Then by Definition 2.2.1, we obtain $t(Ay) - A\bar{s} \in C$. Since $C \subseteq \mathcal{R}^p_{\geqq}$, we obtain $t(Ay) - A\bar{s} \in \mathcal{R}^p_{\geqq}$ and hence, again by Definition 2.2.1, we have that $A\bar{s} \leqq_{\mathcal{R}^p_{\geqq}} t(Ay) = t(u)$. Since u is arbitrary, for all $u \in N(A[Y], \mathcal{R}^p_{\geqq})$ there exists $s \in S$ such that $s \leqq_{\mathcal{R}^p_{\geqq}} t(u)$. Thus the proof is complete. □

Proposition 2.3.5. *Let Y be a set in $\mathcal{R}^p$ and C be a polyhedral set in $\mathcal{R}^p$, $C = \{d \in \mathcal{R}^p : Ad \geqq b\}$ where A is a $p \times p$ matrix and $b \in \mathcal{R}^p$, such that $\mathcal{R}^p_{\geqq b} \subseteq C$. Let $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $b \notin -\mathcal{R}^p_{\geqq}$ and $S \in \mathcal{C}_t(N(A[Y], \mathcal{R}^p_{\geqq b}), \mathcal{R}^p_{\geqq b})$ then $S \in \mathcal{C}_t(A[N(Y, C)], C)$.*

Proof. Let $S \in \mathcal{C}_t(N(A[Y], \mathcal{R}^p_{\geqq b}), \mathcal{R}^p_{\geqq b})$. Since C is a polyhedral cone and $b \notin -\mathcal{R}^p_{\geqq}$, $N(A[Y], \mathcal{R}^p_{\geqq b}) \subseteq A[N(Y, C)]$, ([28], Proposition 4.3) and we obtain $S \subseteq A[N(Y, C)]$. Let $u \in A[N(Y, C)]$. By Definition 2.2.7, there exists $s \in S$ such that $s \leqq_{\mathcal{R}^p_{\geqq b}} t(u)$. Then by Definition 2.2.1, $t(u) - s \in \mathcal{R}^p_{\geqq b}$, and since $\mathcal{R}^p_{\geqq b} \subseteq C$, $t(u) - s \in C$. Again by Definition 2.2.1, we have that $s \leqq_C t(u)$. Since u is arbitrary, for all $u \in A[N(Y, C)]$ there exists $s \in S$ such that $s \leqq_C t(u)$. Thus the proof is complete. □

Proposition 2.3.6. *Let Y be a set in $\mathcal{R}^p$ and C be a polyhedral set in $\mathcal{R}^p$, $C = \{d \in \mathcal{R}^p : Ad \geqq b\}$ where A is a $p \times p$ matrix and $b \in \mathcal{R}^p$, such that $C \subseteq \mathcal{R}^p_{\geqq b}$. Let $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S \in \mathcal{C}_t(A[N(Y, C)], C)$ then $S \in \mathcal{C}_t(N(A[Y], \mathcal{R}^p_{\geqq b}), \mathcal{R}^p_{\geqq b})$.*

Proof. Let $S \in \mathcal{C}_t(A[N(Y, C)], C)$ with $S = A[\bar{S}]$ for some $\bar{S} \subseteq N(Y, C)$. Since C is a polyhedral set, $A[N(Y, C)] \subseteq N(A[Y], \mathcal{R}^p_{\geqq b})$ and we obtain $S \subseteq N(A[Y], \mathcal{R}^p_{\geqq b})$. Let $u \in N(A[Y], \mathcal{R}^p_{\geqq b})$ with $u = Ay$ for some $y \in Y$. By Definition 2.2.7, there exists $s \in S$ with $s = A\bar{s}$ for some $\bar{s} \in Y$ such that $s \leqq_C t(u)$ or, equivalently, $A\bar{s} \leqq_C t(Ay)$. Then by Definition 2.2.1, we obtain $t(Ay) - A\bar{s} \in C$. Since $C \subseteq \mathcal{R}^p_{\geqq b}$, we obtain $t(Ay) - A\bar{s} \in \mathcal{R}^p_{\geqq b}$ and hence, again by Definition 2.2.1, we have that $A\bar{s} \leqq_{\mathcal{R}^p_{\geqq b}} t(Ay) = t(u)$. Since u is arbitrary, for all $u \in N(A[Y], \mathcal{R}^p_{\geqq b})$ there exists $s \in S$ such that $s \leqq_{\mathcal{R}^p_{\geqq b}} t(u)$. Thus the proof is complete. □

We now focus on covers with respect to general cones. We consider covers of multiple sets with respect to multiple cones and multiple tolerance functions. We show an application of multiple sets in Section 6. Multiple cones may model changing or different preferences of the decision makers engaged in the decision making process.

Given a cover and two sets $Y_1, Y_2 \subset \mathcal{R}^p$, a condition on these sets implies that the cover works for both sets.

Proposition 2.3.7. *Let Y_1, Y_2 be two sets in $\mathcal{R}^p$ such that $Y_2 \subseteq Y_1$, C be a cone in $\mathcal{R}^p$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S \in \mathcal{C}_t(Y_1, C)$ then $S \in \mathcal{C}_t(Y_2, C)$.*

Proof. Let $S \in \mathcal{C}_t(Y_1, C)$. By Definition 2.2.7, for all $y \in Y_1$ there exists $s \in S$ such that $s \leqq_C t(y)$. Since $Y_2 \subseteq Y_1$, for all $y \in Y_2$ there exists $s \in S$ such that $s \leqq_C t(y)$. Hence $S \in \mathcal{C}_t(Y_2, C)$. □

Similar to the previous proposition, given a cover of a set and two cones, a condition on the cones implies that the cover works for the set and both cones.

Proposition 2.3.8. *Let Y be a set in $\mathcal{R}^p$, C_1, C_2 be two cones in $\mathcal{R}^p$ such that $C_2 \subseteq C_1$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S \in \mathcal{C}_t(Y, C_2)$ then $S \in \mathcal{C}_t(Y, C_1)$.*

Proof. Let $S \in \mathcal{C}_t(Y, C_2)$ and $y \in Y$. Then by Definition 2.2.7, there exists $s \in S$ such that $s \leqq_{C_2} t(y)$, or equivalently, $t(y) - s \in C_2$. Since $C_2 \subseteq C_1$, we get $t(y) - s \in C_1$. Thus we obtain $s \leqq_{C_1} t(y)$ for some $s \in S$. Since y is arbitrary, for all $y \in Y$ there exists $s \in S$ such that $s \leqq_{C_1} t(y)$ and hence $S \in \mathcal{C}_t(Y, C_1)$, which completes the proof. □

The next proposition is given without a proof.

Proposition 2.3.9. *Let Y be a set in $\mathcal{R}^p$, C_1, C_2 be two cones in $\mathcal{R}^p$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S \in \mathcal{C}_t(Y, C_1 \cap C_2)$ then $S \in \mathcal{C}_t(Y, C_1)$ and $S \in \mathcal{C}_t(Y, C_2)$.*

Based on Propositions 2.3.7 and 2.3.8, the following property holds.

Corollary 2. *Let Y_1, Y_2 be two sets in $\mathcal{R}^p$ such that $Y_2 \subseteq Y_1$, C_1, C_2 be two cones in $\mathcal{R}^p$ such that $C_2 \subseteq C_1$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S \in \mathcal{C}_t(Y_1, C_2)$ then $S \in \mathcal{C}_t(Y_2, C_1)$.*

Proof. By Proposition 2.3.7 we have $S \in \mathcal{C}_t(Y_2, C_2)$, and by Proposition 2.3.8 we have $S \in \mathcal{C}_t(Y_2, C_1)$. $\square$

For algorithmic developments it is important to know how a cover changes when the tolerance function is relaxed. Consider two tolerance functions producing proxy elements dominating each other.

Proposition 2.3.10. *Let Y be a set in $\mathcal{R}^p$ and C be a convex cone in $\mathcal{R}^p$. Let $t_1 : \mathcal{R}^p \mapsto \mathcal{R}^p$ and $t_2 : \mathcal{R}^p \mapsto \mathcal{R}^p$ be two tolerance functions such that $t_1(y) \leqq_C t_2(y)$ for all $y \in Y$. If $S \in \mathcal{C}_{t_1}(Y, C)$ then $S \in \mathcal{C}_{t_2}(Y, C)$.*

Proof. Let $S \in \mathcal{C}_{t_1}(Y, C)$ and $y \in Y$. Then by Definition 2.2.7, there exists $s \in S$ such that $s \leqq_C t_1(y)$ or equivalently $t_1(y) - s \in C$. We have that $t_1(y) \leqq_C t_2(y)$ for all $y \in Y$, or equivalently, $t_2(y) - t_1(y) \in C$ for all $y \in Y$. Since C is a convex cone, we obtain $t_1(y) - s + t_2(y) - t_1(y) \in C$. Thus we get $t_2(y) - s \in C$ and hence $s \leqq_C t_2(y)$. Since y is arbitrary, for all $y \in Y$ there exists $s \in S$ such that $s \leqq_C t_2(y)$. $\square$

We examine the behavior of a cover under the composition of two tolerance functions.

Proposition 2.3.11. *Let Y be a set in $\mathcal{R}^p$ and C be a convex cone in $\mathcal{R}^p$. Let $t_1 : \mathcal{R}^p \mapsto \mathcal{R}^p$ and $t_2 : \mathcal{R}^p \mapsto \mathcal{R}^p$ be two tolerance functions. If $S \in \mathcal{C}_{t_1}(Y, C)$ then $S \in \mathcal{C}_{t_2(t_1)}(Y, C)$.*

Proof. Let $S \in \mathcal{C}_{t_1}(Y, C)$ and $y \in Y$. Then by Definition 2.2.7, there exists $s \in S$ such that $s \leqq_C t_1(y)$. Since t_2 is a tolerance function, by Definition 2.2.5, we get $t_1(y) \leqq_C t_2(t_1(y))$. Since C is a convex cone, $t_2(t_1(y)) - t_1(y) + t_1(y) - s \in C$. Thus we get $t_2(t_1(y)) - s \in C$ and hence $s \leqq_C t_2(t_1(y))$. Since y is arbitrary, for all $y \in Y$ there exists $s \in S$ such that $s \leqq_C t_2(t_1(y))$. $\square$

Proposition 2.3.7 together with Proposition 2.3.10 yield the following result.

Corollary 3. *Let Y_1, Y_2 be two sets in $\mathcal{R}^p$ such that $Y_2 \subseteq Y_1$ and C be a convex cone in $\mathcal{R}^p$. Let $t_1 : \mathcal{R}^p \mapsto \mathcal{R}^p$ and $t_2 : \mathcal{R}^p \mapsto \mathcal{R}^p$ be two tolerance functions such that $t_1(y) \leqq_C t_2(y)$ for all $y \in Y$. If $S \in \mathcal{C}_{t_1}(Y_1, C)$ then $S \in \mathcal{C}_{t_2}(Y_2, C)$.*

Proof. By Proposition 2.3.7, we have $S \in \mathcal{C}_{t_1}(Y_2, C)$. Since C is a convex cone, by Proposition 2.3.10, we have $S \in \mathcal{C}_{t_2}(Y_2, C)$. $\square$

Proposition 2.3.8 together with Proposition 2.3.10 yield the following result.

Corollary 4. *Let Y be a set in $\mathcal{R}^p$ and C_1, C_2 be two cones in $\mathcal{R}^p$ such that $C_2 \subseteq C_1$ and C_1 be convex. Let $t_1 : \mathcal{R}^p \mapsto \mathcal{R}^p$ and $t_2 : \mathcal{R}^p \mapsto \mathcal{R}^p$ be two tolerance functions such that $t_1(y) \leqq_{C_1} t_2(y)$ for all $y \in Y$. If $S \in \mathcal{C}_{t_1}(Y, C_2)$ then $S \in \mathcal{C}_{t_2}(Y, C_1)$.*

Proof. By Proposition 2.3.8, we have $S \in \mathcal{C}_{t_1}(Y, C_1)$. Since C_1 is a convex cone, by Proposition 2.3.10, we have $S \in \mathcal{C}_{t_2}(Y, C_1)$. $\square$

The next three propositions present properties of covers for sets composed of subsets that are combined using the set operations such as the Cartesian product, algebraic sum, and union. Each result involves two sets, two cones, and two tolerance functions and can be generalized to a bigger number of these items.

Proposition 2.3.12. *Let Y_i be a set in $\mathcal{R}^{p_i}$, C_i be a cone in $\mathcal{R}^{p_i}$, and $t_i : \mathcal{R}^{p_i} \mapsto \mathcal{R}^{p_i}$ be a tolerance function for $i = 1, 2$. Then $S_1 \in \mathcal{C}_{t_1}(Y_1, C_1)$ and $S_2 \in \mathcal{C}_{t_2}(Y_2, C_2)$ if and only if $S_1 \times S_2 \in \mathcal{C}_t(Y_1 \times Y_2, C_1 \times C_2)$, where $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ is defined as $t(y_1, y_2) = (t_1(y_1), t_2(y_2))$ for $y_1 \in Y_1$, and $y_2 \in Y_2$, and $p = p_1 + p_2$.*

Proof. Let $S_1 \in \mathcal{C}_{t_1}(Y_1, C_1)$ and $S_2 \in \mathcal{C}_{t_2}(Y_2, C_2)$. By Definition 2.2.7, for all $y_i \in Y_i$ there exists $s_i \in S_i$ such that $s_i \leqq_{C_i} t_i(y_i), i = 1, 2$. These relations hold if and only if for all $(y_1, y_2) \in Y_1 \times Y_2$ there exists $(s_1, s_2) \in S_1 \times S_2$ such that $t_1(y_1) - s_1 \in C_1$ and $t_2(y_2) - s_2 \in C_2$ or, equivalently, $(t_1(y_1), t_2(y_2)) - (s_1, s_2) \in C_1 \times C_2$. That is, for all $(y_1, y_1) \in Y_1 \times Y_2$, there exists $(s_1, s_2) \in S_1 \times S_2$ such that $(s_1, s_2) \leqq_{C_1 \times C_2} (t_1(y_1), t_2(y_2))$ or, equivalently, $(s_1, s_2) \leqq_{C_1 \times C_2} t(y_1, y_2)$ with $t = (t_1, t_2)$. $\square$

The next proposition requires that the emerging tolerance function be additively separable.

Proposition 2.3.13. *Let Y_i be a set in $\mathcal{R}^p$, C_i be a cone in $\mathcal{R}^p$, and $t_i : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function for $i = 1, 2$. If $S_1 \in \mathcal{C}_{t_1}(Y_1, C_1)$ and $S_2 \in \mathcal{C}_{t_2}(Y_2, C_2)$ then $S_1 + S_2 \in \mathcal{C}_t(Y_1 + Y_2, C_1 + C_2)$, where $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ is defined as $t(y_1 + y_2) = t_1(y_1) + t_2(y_2)$ for $y_1 \in Y_1$ and $y_2 \in Y_2$.*

Proof. Let $S_1 \in \mathcal{C}_{t_1}(Y_1, C_1)$ and $S_2 \in \mathcal{C}_{t_2}(Y_2, C_2)$. Let $y_1 \in Y_1$ and $y_2 \in Y_2$. Then by Definition 2.2.7, there exists $s_1 \in S_1$ such that $s_1 \leqq_{C_1} t_1(y_1)$ and there exists $s_2 \in S_2$ such that $s_2 \leqq_{C_2} t_2(y_2)$, which implies that for $y_1 + y_2 \in Y_1 + Y_2$ there exists $s_1 + s_2 \in S_1 + S_2$ such that $t_1(y_1) - s_1 \in C_1$ and $t_2(y_2) - s_2 \in C_2$. Thus we have $t_1(y_1) - s_1 + t_2(y_2) - s_2 \in C_1 + C_2$. That is, $s_1 + s_2 \leqq_{C_1 + C_2} t_1(y_1) + t_2(y_2)$, or equivalently, $s_1 + s_2 \leqq_{C_1 + C_2} t(y_1 + y_2)$ due to the definition of t. Since y_1 and y_2 are arbitrary, we obtain for all $y_1 + y_2 \in Y_1 + Y_2$ there exists $s_1 + s_2 \in S_1 + S_2$ such that $s_1 + s_2 \leqq_{C_1 + C_2} t(y_1 + y_2)$. □

The final proposition of this section again makes use of two tolerance functions that produce proxy elements dominating each other.

Proposition 2.3.14. *Let Y_1, Y_2 be two sets in $\mathcal{R}^p$ and C_1, C_2 be two cones in $\mathcal{R}^p$ such that C_1 is convex. Let $t_1, t_2 : \mathcal{R}^p \mapsto \mathcal{R}^p$ be tolerance functions such that $t_1(y_1) \leqq_{C_1} t_2(y_1)$ for all $y_1 \in Y_1$. If $S_1 \in \mathcal{C}_{t_1}(Y_1, C_1)$ and $S_2 \in \mathcal{C}_{t_2}(Y_2, C_2)$ then $S_1 \cup S_2 \in \mathcal{C}_{t_2}(Y_1 \cup Y_2, C_1 \cup C_2)$.*

Proof. Let $S_1 \in \mathcal{C}_{t_1}(Y_1, C_1)$ and $S_2 \in \mathcal{C}_{t_2}(Y_2, C_2)$. Let $y_1 \in Y_1$. Then by Definition 2.2.7, there exists $s_1 \in S_1$ such that $s_1 \leqq_{C_1} t_1(y_1)$, or equivalently $t_1(y_1) - s_1 \in C_1$. Since $t_1(y_1) \leqq_{C_1} t_2(y_1)$, we obtain $t_2(y_1) - t_1(y_1) \in C_1$. Since C_1 is convex, $t_1(y_1) - s_1 + t_2(y_1) - t_1(y_1) \in C_1$. Thus, we have $t_2(y_1) - s_1 \in C_1$. Now, let $y_2 \in Y_2$. Then by Definition 2.2.7, there exists $s_2 \in S_2$ such that $s_2 \leqq_{C_2} t_2(y_2)$, or equivalently, $t_2(y_2) - s_2 \in C_2$. This implies that for any $y \in Y_1 \cup Y_2$, there exists $s \in S_1 \cup S_2$ such that $t_2(y) - s \in C_1 \cup C_2$. Since $y_1 \in Y_1$ and $y_2 \in Y_2$ are arbitrary, for all $y \in Y_1 \cup Y_2$ there exists $s \in S_1 \cup S_2$ such that $s \leqq_{C_1 \cup C_2} t_2(y)$. □

2.4 Properties of approximations

In this section we present additional properties that are relevant only for approximations. We begin with the following obvious property.

Corollary 5. *If $S \in \mathcal{A}_t(Y, C)$ then $S \in \mathcal{C}_t(Y, C)$.*

Proposition 2.4.1 addresses the construction of an approximation set from a cover. Filtering a t-cover by removing dominated elements provides a t-approximation set. The cover and the filtering do not have to rely on the same cone.

Proposition 2.4.1. *Let Y be a set in $\mathcal{R}^p$, C_1, C_2 be two cones in $\mathcal{R}^p$ such that $C_2 \subseteq C_1$, and C_1 be a convex cone. Let $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S \in \mathcal{C}_t(Y, C_1)$ then $N(S, C_2) \in \mathcal{A}_t(Y, C_1)$.*

Proof. We need to show that (1) $N(S, C_2) \in \mathcal{C}_t(Y, C_1)$ and (2) $N(N(S, C_2), C_1) = N(S, C_2)$.
(1): Let $s \in S \backslash N(S, C_2)$. Since $s \notin N(S, C_2)$, there exists $s' \in N(S, C_2)$ such that $s' \leq_{C_2} s$. Let $y \in Y$ such that $s \leqq_{C_1} t(y)$ for $s \in S$. Since $s' \leq_{C_2} s$ and $s \leqq_{C_1} t(y)$ then there exists $d_2 \in C_2, d_2 \neq 0$, such that $s' + d_2 = s$ and $d_1 \in C_1$ such that $s + d_1 = t(y)$. Since $C_2 \subseteq C_1$ we get $d_2 \in C_1$. As C_1 is a convex cone, there exists $d = d_1 + d_2 \in C_1$ such that $s' + d = t(y)$, and $s' \leqq_{C_1} t(y)$. Since y is covered by $s' \in N(S, C_2)$ with respect to the cone C_1 s can be removed from the cover S. Since s is arbitrary, repeating this process for all $s \in S \setminus N(S, C_2)$ we obtain that all $s \in S \setminus N(S, C_2)$ such that $s \leqq_{C_1} t(y)$ for $y \in Y$ can be removed from the cover S and y can be covered by $s' \in N(S, C_2)$ with respect to C_1. Therefore, $N(S, C_2) \in \mathcal{C}_t(Y, C_1)$.

(2): Let $s \in N(S, C_2) \setminus N(N(S, C_2), C_1)$. Since $s \notin N(N(S, C_2), C_1)$, there exists $s' \in N(N(S, C_2), C_1)$ such that $s' \leq_{C_1} s$. Let $y \in Y$ such that $s \leqq_{C_1} t(y)$ for $s \in N(S, C_2) \setminus N(N(S, C_2), C_1)$. Since $s' \leq_{C_1} s$ and $s \leqq_{C_1} t(y)$ then there exists $d_1 \in C_1, d_1 \neq 0$, such that $s' + d_1 = s$ and $d_1' \in C_1$ such that $s + d_1' = t(y)$. As C_1 is a convex cone, there exists $d = d_1 + d_1' \in C_1$ such that $s' + d = t(y)$, and $s' \leqq_{C_1} t(y)$. Since y is covered by $s' \in N(N(S, C_2), C_1)$ with respect to the cone C_1, s can be removed from the cover $N(S, C_2)$. Since s is arbitrary, repeating this process for all $s \in N(S, C_2) \setminus N(N(S, C_2), C_1)$ we obtain that all $s \in N(S, C_2) \setminus N(N(S, C_2), C_1)$ such that $s \leqq_{C_1} t(y)$ for $y \in Y$ can be removed from the cover $N(S, C_2)$ and y can be covered by $s' \in N(N(S, C_2), C_1)$ with respect to C_1. That is, $N(S, C_2)$ has no dominated points with respect to the cone C_1 and $N(S, C_2) = N(N(S, C_2), C_1)$.

Therefore by **(1)** and **(2)**, we obtain $N(S, C_2) \in \mathcal{A}_t(Y, C_1)$

□

Relying on the same cone in Proposition 2.4.1, we obtain the following corollary.

Corollary 6. *Let Y be a set in $\mathcal{R}^p$, C be a convex cone in $\mathcal{R}^p$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S \in \mathcal{C}_t(Y, C)$ then $N(S, C) \in \mathcal{A}_t(Y, C)$.*

In view of the two results above, the reduction of a cover to an approximation set is possible with a convex cone even that covers are defined with respect to general cones.

The approximation is considered perfect when it does not allow any tolerance, which is modeled with the identity function. In this case, the set of all approximations reduces to the nondominated set.

Proposition 2.4.2. *Let Y be a set in $\mathcal{R}^p$ and C be a cone in $\mathcal{R}^p$. Let $id : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function such that $id(y) = y$ for all $y \in Y$. Then $\mathcal{A}_{id}(Y, C) = \{N(Y, C)\}$.*

Proof. Let $S \in \mathcal{C}_t(Y, C)$ for $t = id$. By Definition 2.2.7, for every $y \in Y$ there exists $s \in S$ such that $s \leqq_C id(y) = y$. Then also for every $y \in N(Y, C)$ there exists $s \in S$ such that $s \leqq_C id(y) = y$. By definition, $y \in N(Y, C)$ if and only if there does not exist $y' \in Y$ such that $y' \leq_C y$. Since $S \subseteq Y$, it must be that $s = y$ with respect to the cone C. Therefore, we have $S = N(Y, C)$ and $N(Y, C) \in \mathcal{A}_{id}(Y, C)$. Thus for any arbitrary cover S, we obtain $S = N(Y, C)$ and hence $\mathcal{A}_{id}(Y, C) = \{N(Y, C)\}$. □

Proposition 2.4.3 shows that given an approximation set S with respect to a polyhedral cone, the image of S under the linear mapping induced by the matrix of the cone, $A[S]$, is an inherently nondominated set with respect to the Pareto cone.

Proposition 2.4.3. *Let Y be a set in $\mathcal{R}^p$ and C be a polyhedral cone in $\mathcal{R}^p$, $C = \{d \in \mathcal{R}^p : \ Ad \geqq 0\}$ where A is a $p \times p$ matrix. Let $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S \in \mathcal{A}_t(Y, C)$ then $N(A[S], \mathcal{R}^p_{\geqq}) = A[S]$.*

Proof. Clearly $N(A[S], \mathcal{R}^p_{\geqq}) \subseteq A[S]$. We show $A[S] \subseteq N(A[S], \mathcal{R}^p_{\geqq})$. By contradiction, let $s \in A[S] \setminus N(A[S], \mathcal{R}^p_{\geqq})$ with $s = As'$ for $s' \in S$. That is, there exists $\bar{s} \in A[S]$ with $\bar{s} = As''$ for $s'' \in S$ and $d \in \mathcal{R}^p_{\geqq}$, $d \neq 0$, such that $s = \bar{s} + d$. We obtain $As' = As'' + d$ and $A(s' - s'') = d \geq 0$ as $d \neq 0$. Therefore, $s' - s'' \in C$ and there exists $d' \in C$, $d' \neq 0$, such that $d' = s' - s''$, or equivalently, $s' = d' + s''$. That is, there exist $s'' \in S$ and $d' \in C$, $d' \neq 0$, such that $s' = s'' + d'$. Thus, $s'' \leq_C s'$ and $N(S, C) \neq S$, which contradicts that $N(S, C) = S$ and proves the result. $\square$

Proposition 2.4.4. *Let Y be a set in $\mathcal{R}^p$ and C be a polyhedral cone in $\mathcal{R}^p$, $C = \{d \in \mathcal{R}^p : Ad \geqq 0\}$ where A is a $p \times p$ matrix. Let $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be an A-invariant tolerance function. If $S \in \mathcal{A}_t(Y, C)$ then $A[S] \in \mathcal{A}_t(A[Y], \mathcal{R}^p_{\geqq})$.*

Proof. Let $S \in \mathcal{A}_t(Y, C)$. By Corollary 5, $S \in \mathcal{C}_t(Y, C)$ and by Proposition 2.3.1, $A[S] \in \mathcal{C}_t(A[Y], \mathcal{R}^p_{\geqq})$. Also, since $S \in \mathcal{A}_t(Y, C)$, by Proposition 2.4.3, we have $N(A[S], \mathcal{R}^p_{\geqq}) = A[S]$. Using Definition 2.2.8, we obtain $A[S] \in \mathcal{A}_t(A[Y], \mathcal{R}^p_{\geqq})$. $\square$

For a proof of Corollary 7 we refer the reader to [37].

Corollary 7. *Let Y_i be a set in $\mathcal{R}^{p_i}$ and C_i be a cone $\mathcal{R}^{p_i}$ for $i = 1, 2$. Then $N(Y_1 \times Y_2, C_1 \times C_2) = N(Y_1, C_1) \times N(Y_2, C_2)$.*

Corollaries 8 and 9 follow on Propositions 2.3.12 and 2.3.13.

Corollary 8. *Let Y_i be a set in $\mathcal{R}^{p_i}$, C_i be a convex cone in $\mathcal{R}^{p_i}$, and $t_i : \mathcal{R}^{p_i} \mapsto \mathcal{R}^{p_i}$ be a tolerance function for $i = 1, 2$. Let $S_i \in \mathcal{C}_{t_i}(Y_i, C_i)$ for $i = 1, 2$. Then $N(S_1, C_1) \in$*

$\mathcal{A}_{t_1}(Y_1, C_1)$ and $N(S_2, C_2) \in \mathcal{A}_{t_2}(Y_2, C_2)$ if and only if $N(S_1 \times S_2, C_1 \times C_2) \in \mathcal{A}_t(Y_1 \times Y_2, C_1 \times C_2)$, where $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ is defined as $t(y_1, y_2) = (t_1(y_1), t_2(y_2))$ for $y_1 \in Y_1$, and $y_2 \in Y_2$, and $p = p_1 + p_2$.

Proof. Since C_i is a convex cone, by Corollaries 6 and 5, $N(S_i, C_i) \in \mathcal{C}_{t_i}(Y_i, C_i)$ for $i = 1, 2$. Thus, by Corollary 7 and Proposition 2.3.12, we obtain the desired result. □

Corollary 9. *Let Y_i be a set in $\mathcal{R}^p$, C_i be a convex cone in $\mathcal{R}^p$, and $t_i : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function for $i = 1, 2$. If $S_1 \in \mathcal{C}_{t_1}(Y_1, C_1)$ and $S_2 \in \mathcal{C}_{t_2}(Y_2, C_2)$ then $N(N(S_1, C_1) + N(S_2, C_2), C_1 + C_2) \in \mathcal{A}_t(Y_1 + Y_2, C_1 + C_2)$, where $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ is defined as $t(y_1 + y_2) = t_1(y_1) + t_2(y_2)$ for $y_1 \in Y_1$ and $y_2 \in Y_2$.*

Proof. Since C_1, C_2 are convex cones, by Corollaries 6 and 5, $N(S_i, C_i) \in \mathcal{C}_{t_i}(Y_i, C_i)$ for $i = 1, 2$. Thus, by Proposition 2.3.13, $N(S_1, C_1) + N(S_2, C_2) \in \mathcal{C}_t(Y_1 + Y_1, C_1 + C_2)$, where $t(y_1 + y_2) = t_1(y_1) + t_2(y_2)$ for $y_1 \in Y_1$ and $y_2 \in Y_2$. If C_1 and C_2 are convex cones, then $C_1 + C_2$ is a convex cone. Thus, again by Corollary 6, we have $N(N(S_1, C_1) + N(S_2, C_2), C_1 + C_2) \in \mathcal{A}_t(Y_1 + Y_2, C_1 + C_2)$. □

Corollaries 8 and 9 specialize Propositions 2.3.12 and 2.3.13 for the approximation sets. Proposition 2.3.14 is applied to approximation sets in a different way, which is reflected in Corollaries 10 and 11.

Corollary 10. *Let Y_1, Y_2 be two sets in $\mathcal{R}^p$, C_1, C_2 be two cones in $\mathcal{R}^p$ such that C_1 and $C_1 \cup C_2$ are convex. Let $t_1, t_2 : \mathcal{R}^p \mapsto \mathcal{R}^p$ be two tolerance functions such that $t_1(y_1) \leqq_{C_1} t_2(y_1)$ for all $y_1 \in Y_1$. If $S_1 \in \mathcal{C}_{t_1}(Y_1, C_1)$ and $S_2 \in \mathcal{C}_{t_2}(Y_2, C_2)$ then $N(S_1 \cup S_2, C_1 \cup C_2) \in \mathcal{A}_t(Y_1 \cup Y_2, C_1 \cup C_2)$.*

Proof. By Proposition 2.3.14, we have $S_1 \cup S_2 \in \mathcal{C}_{t_2}(Y_1 \cup Y_2, C_1 \cup C_2)$. Since $C_1 \cup C_2$ is convex, by Corollary 6, we have the desired result. □

Corollary 11. *Let Y_1, Y_2 be two sets in $\mathcal{R}^p$, C be a cone in $\mathcal{R}^p$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. If $S_1 \in \mathcal{C}_t(Y_1, C)$ and $S_2 \in \mathcal{C}_t(Y_2, C)$ then $N(S_1 \cup S_2, C) \in \mathcal{A}_t(Y_1 \cap Y_2, C)$.*

Proof. By Proposition 2.3.14, we have $S_1 \cup S_2 \in \mathcal{C}_t(Y_1 \cup Y_2, C)$. Since $Y_1 \cap Y_2 \subseteq Y_1 \cup Y_2$, by Proposition 2.3.7, we have that $S_1 \cup S_2 \in \mathcal{C}_t(Y_1 \cap Y_2, C)$. Since C is a convex cone, by Corollary 6, we obtain the desired result. $\square$

2.5 Examples and application

We have investigated the properties of covers and approximations of solution sets which are defined based on the concept of a tolerance function. These properties are applicable to the various approximating sets produced by the algorithms available in the literature. These algorithm yield covers and approximations, as defined in this paper or other approximation sets. Although the tolerance function implied by those algorithms is not formally recognized by the authors, it is available. Information about some of those research efforts is contained in Table 2.1. The first column on the left of this table gives the type of the MOP for which an approximation algorithm has been designed, the second column displays the number of objective functions of this MOP, and the third column shows the tolerance function implied by the algorithm. The most right column contains the corresponding reference.

In this section we also discuss further applicability of our results. Consider complex decision making problems that are modeled as collections of MOPs. Since the calculations of their nondominated sets is even more challenging due to interactions between component MOPs, the overall complex problem has to be decomposed into the component MOPs that are more easily solved ([38], [39]). However, as explained in Section 2.1, even for a decomposed problem, it is often hard to obtain exact solutions to the subproblems and so the solution sets have to be approximated separately before the overall approximation can be constructed. The goal is then to approximate the nondominated set of the overall system by approximating the nondominated sets of the subproblems.

Let the Pareto set of the all-in-one (AiO) problem be denoted by $N(Y, \mathcal{R}^p_{\geqq})$, where

problem	p	$t : \mathcal{R}^p \mapsto \mathcal{R}^p$	reference
web sources access	2	$(1+\epsilon)y$	Papadimitriou and Yannakakis (2000)
shortest path	2	$2y$	Diakonikolas and Yannakakis (2001)
spanning tree	2	$2y$	
knapsack	2	$2y$	
scheduling	2	$2y$	
knapsack	≥ 2	$(1+\epsilon)y$	Erlebach et al. (2002)
	≥ 2	$(1+\epsilon)y$	Bazgan et al. (2009)
traveling salesman	2	$1.5y$	Angel et al. (2004)
traveling salesman	2	$1.5y$	Angel et al. (2005)
	≥ 3	$\frac{2p}{(p+1)}y$	
traveling salesman	≥ 2	$\min\{1+\gamma, \alpha+\epsilon)\}y$, for $\alpha = \frac{2\gamma^2}{(2\gamma^2-2\gamma+1)}$	Manthey and Ram (2009)
MOP	≥ 2	$y+\epsilon$	Legriel et al. (2010)
MOP	≥ 2	$y+\epsilon$	Laumanns and Zenklusen (2011)
MOCO	2	$(1+\epsilon)y$	Filippi and Stevanato (2013)

Table 2.1: Examples of tolerance functions

the set of outcomes $Y \subseteq \mathcal{R}^p$, $Y = f(X)$, is the image the set of feasible decisions $X \subseteq \mathcal{R}^n$ with the vector-valued objective function $f : \mathcal{R}^n \to \mathcal{R}^p$. Depending upon the properties of the feasible set X and the function f, three types of decomposition of the AiO problem are presented which lead to three different calculations of $N(Y, \mathcal{R}^p_{\geqq})$.

1. Let the feasible set X be partitioned into two subsets, $X = X_1 \times X_2$, $X_i \subseteq \mathcal{R}^{n_i}, i = 1, 2$, and $n_1 + n_2 = n$.

   (a) Assume that the function f is composed of functions $f_i : \mathcal{R}^{n_i} \to \mathcal{R}^{p_i}$, $f(x) = (f_1(x_1), f_2(x_2))$, where $p_1+p_2 = p$. Then $Y = Y_1 \times Y_2$, where $Y_i = f_i(X_i), i = 1, 2$. Applying Proposition 3.9 of [38], we obtain

$$N(Y_1 \times Y_2, \mathcal{R}^{p_1}_{\geqq} \times \mathcal{R}^{p_2}_{\geqq}) = N(Y_1, \mathcal{R}^{p_1}_{\geqq}) \times N(Y_2, \mathcal{R}^{p_2}_{\geqq}).$$

Given a cover or an approximation set for $N(Y_i, \mathcal{R}^{p_i}_{\geqq}), i = 1, 2$, and using Proposition 2.3.12 or Corollary 8, a cover or an approximation set of $N(Y_1 \times Y_2, \mathcal{R}^{p_1}_{\geqq} \times \mathcal{R}^{p_2}_{\geqq})$ for the AiO problem can be constructed.

(b) Under the assumption that the function f is additively separable, ($f(x) = f_1(x_1) + f_2(x_2)$, where $f_i : \mathcal{R}^{n_i} \to \mathcal{R}^p, i = 1, 2$), we have $Y = Y_1 + Y_2$, where $Y_i = f_i(X_i), i = 1, 2$. Applying Proposition 3.3 of [38], we obtain

$$N(Y_1 + Y_2, \mathcal{R}^p_{\geqq}) = N(N(Y_1, \mathcal{R}^p_{\geqq}) + N(Y_2, \mathcal{R}^p_{\geqq}), \mathcal{R}^p_{\geqq}).$$

Given a cover or an approximation set of $N(Y_i, \mathcal{R}^p_{\geqq}), i = 1, 2$, by applying Proposition 2.3.13 or Corollary 9, a cover or an approximation set of $N(Y_1 + Y_2, \mathcal{R}^p_{\geqq})$ for the AiO problem can be constructed. Note that additive separability is a property commonly found in engineering applications ([42]).

2. Let the feasible set X be partitioned into two subsets, $X = X_1 \cup X_2$, $X_i \subseteq \mathcal{R}^n, i = 1, 2$, and the function f be additively separable. Then $Y = Y_1 \cup Y_2$, where $Y_i = f_i(X_i), i = 1, 2$. Extending Proposition 3.2 of [38], we obtain the following result

$$N\big(Y_1 \cup Y_2, \mathcal{R}^p_{\geqq}\big) = N\big(N(Y_1, \mathcal{R}^p_{\geqq}) \cup N(Y_2, \mathcal{R}^p_{\geqq}), \mathcal{R}^p_{\geqq}\big).$$

Given a cover or an approximation set for $(N(Y_i, \mathcal{R}^p_{\geqq}), \mathcal{R}^p_{\geqq})$, $i = 1, 2$ and applying Proposition 2.3.14 or Corollary 10, a cover or an approximation set of $N\big(Y_1 \cup Y_2, \mathcal{R}^p_{\geqq}\big)$ for the AiO problem can be constructed. For example, this case is applicable to mixed-integer MOPs whose feasible region can be decomposed into the union of subsets by fixing the integer variables to their feasible values ([9]).

The results presented above can obviously be generalized for more than two subproblems and for general convex cones C in place of the Pareto cone $\mathcal{R}^p_{\geqq}$.

2.6 t-dominance and conclusion

The proposed tolerance function leads to a definition of tolerance-based dominance or t-dominance for MOPs. In this section we investigate this concept and its properties.

Let Y be a set in $\mathcal{R}^P$, $y \in Y$ and $d \in \mathcal{R}^P$. According to Definition 2.2.2, if $y \leqq_C y+d$, then the vector d is called a domination direction at y. Now let $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. We can define t-dominance for $y \in Y$ in two different ways: (i) $y \leqq_C t(y+d)$ or (ii) $t(y) \leqq_C y+d$. Consider case (i). Since y dominates $y+d$, and by Definition 2.2.5, $y+d$ dominates $t(y+d)$, then case (i) holds for every $y \in Y$ and is trivial.

Consider case (ii). Again, by Definition 2.2.5, $y \leqq_C t(y)$ and hence $y \leqq_C t(y) \leqq_C y+d$, which means the tolerance function allows the user to relax y but maintain its dominance over $y+d$. We therefore define t-dominance as follows:

Definition 2.6.1. *Let Y be a set in $\mathcal{R}^p$, C be a cone in $\mathcal{R}^p$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. A point $y' \in Y$ is called a t-dominated point of the set Y with respect to the cone C and the tolerance function t if there exists a point $y \in Y$ such that $t(y) \leqq_C y'$.*

Definition 2.6.2. *Let Y be a set in $\mathcal{R}^p$, C be a cone in $\mathcal{R}^p$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function.*

1. *A vector $d \in \mathcal{R}^p$ is said to be a domination direction of $y \in Y$ with respect to the cone C and the tolerance function t if $t(y) \leqq_C y+d$.*

2. *The set of all domination directions of $y \in Y$ with respect to C and t is defined as $D_t(y) = \{d \in \mathcal{R}^p : t(y) \leqq_C y+d\}$.*

3. *The set $\mathcal{D}_t = \{D_t(y) : y \in Y\}$ is called the domination set for Y.*

In this section we assume that $D_t(y)$ is a nonempty set for all $y \in Y$. Following [63], we define a translated cone and show that the set $D_t(y)$ is such a cone.

Definition 2.6.3. *Let C be a cone in $\mathcal{R}^p$ and v be a vector in $\mathcal{R}^p$. A cone with vertex v is defined as a translation $v + C$ of the cone C.*

Corollary 12. *Let C be a cone in $\mathcal{R}^p_{\geqq}$. The set $D_t(y) = \{d \in \mathcal{R}^p : t(y) \leqq_C y + d\}$ is a translated cone.*

Proof. Let $d \in D_t(y)$ and show that $\lambda d \in D_t(y)$ for $\lambda > 0$. By Definition 2.6.2, $t(y) \leqq_C y+d$ or equivalently $y + d - t(y) \in C$. Thus $d \in t(y) - y + C$, where $t(y) - y + C$ is a translation of the cone C, and it is a cone with vertex $t(y) - y$. Therefore, λd is in the cone C with vertex $t(y) - y$. This completes the proof. $\square$

Remark 2.6.1. *Note that the representation of $D_t(y)$ for $y \in Y$ in Definition 2.6.2 can be written as $D_t(y) = t(y) - y + C$. If C is a convex pointed cone, then $D_t(y)$ is a translated convex, pointed cone with vertex at $t(y) - y$.*

Two tolerance functions given in Example 2.6.1 are commonly used in the literature.

Example 2.6.1. *Let Y be a set in $\mathcal{R}^p$ and C be a convex cone in $\mathcal{R}^p$.*

1. *Let $t(y) = (1 + \epsilon)y$, for $\epsilon \in \mathcal{R} : \epsilon y \in C$ for all $y \in Y$. Then $D_t(y) = \epsilon y + C$.*
2. *Let $t(y) = y + \epsilon$, for $\epsilon \in C$. Then $D_t(y) = \epsilon + C$.*

Given two tolerance functions t_1, t_2, two types of conditions guarantee that $\mathcal{D}_{t_1} \subseteq \mathcal{D}_{t_2}$.

Proposition 2.6.1. *Let Y be a set in $\mathcal{R}^p$ and $t_1, t_2 : \mathcal{R}^p \mapsto \mathcal{R}^p$ be two tolerance functions. If $D_{t_1}(y) \subseteq D_{t_2}(y)$ for all $y \in Y$ then $\mathcal{D}_{t_1} \subseteq \mathcal{D}_{t_2}$.*

Proof. Let $y \in Y$ and $D_{t_1}(y) \in \mathcal{D}_{t_1}$ where $D_{t_1}(y) = \{d \in \mathcal{R}^p : t_1(y) \leqq_C y + d\}$. Let $d \in D_{t_1}(y)$. Since $D_{t_1}(y) \subseteq D_{t_2}(y)$ for all $y \in Y$, we have $d \in D_{t_2}(y) = \{d \in \mathcal{R}^p : t_2(y) \leqq_C y+d\}$, i.e., if d is a domination direction of y with respect to t_1, then it is also a domination direction of y with respect to t_2. Since y is arbitrary, the property holds for all $y \in Y$ and the proof is complete. $\square$

Proposition 2.6.2. *Let Y be a set in $\mathcal{R}^p$, C be a convex cone in $\mathcal{R}^p$, and $t_1, t_2 : \mathcal{R}^p \mapsto \mathcal{R}^p$ be two tolerance functions such that $t_2(y) \leqq_C t_1(y)$ for all $y \in Y$. Then $\mathcal{D}_{t_1} \subseteq \mathcal{D}_{t_2}$.*

Proof. Since $t_2(y) \leqq_C t_1(y)$ we have $t_1(y) - t_2(y) \in C$, or equivalently, $t_1(y) \in C + t_2(y)$. It follows that $t_1(y) + C \subseteq C + C + t_2(y) \subseteq C + t_2(y)$ since C is a convex cone. Now we get $t_1(y) - y + C \subseteq t_2(y) - y + C$. By Remark 2.6.1, $D_{t_i}(y) = t_i(y) - y + C$ for $i = 1, 2$ and hence $D_{t_1}(y) \subseteq D_{t_2}(y)$. Since y is arbitrary we have $D_{t_1}(y) \subseteq D_{t_2}(y)$ for all $y \in Y$ and hence, by Proposition 2.6.1, we obtain $\mathcal{D}_{t_1} \subseteq \mathcal{D}_{t_2}$. $\square$

We now define t-nondominated points with respect to a cone C.

Definition 2.6.4. *Let Y be a set in $\mathcal{R}^p$, C be a convex cone in $\mathcal{R}^p$, and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function. A point $y' \in Y$ is called a t-nondominated point of the set Y with respect to the cone C and the tolerance function t, if there does not exist a point $y \in Y$ such that $t(y) \leq_C y'$. The set of all t-nondominated points of Y with respect to the cone C and the tolerance function t is denoted by $N(Y, C, t)$.*

If $t(y) = y + \epsilon$, $\epsilon \in C \setminus \{0\}$, the concept of t-nondominated points reduces to the concept of ϵ-minimal points in [28].

We obtain the following obvious property.

Corollary 13. *$N(Y, C) \subseteq N(Y, C, t)$ for every tolerance function t.*

Example 2.6.2. *Let Y and t be defined as in Example 2.2.4.*

1. *Let the cone C be defined as in part 1. The points $y^1, y^2, y^4, y^5 \in N(Y, C)$ and, by Corollary 13, these points are in $N(Y, C, t)$. Additionally, there does not exist $y \in Y$ such that $t(y) \leq_C y^3$. Thus $y^3 \in N(Y, C, t)$. However, $y^6 \notin N(Y, C, t)$ because $t(y^4) \leq_C y^6$. Therefore $N(Y, C, t) = \{y^1, y^2, y^3, y^4, y^5\}$.*

2. *Let the cone C be defined as in part 2. The points $y^1, y^2, y^4 \in N(Y, \mathcal{R}^2_{\geqq})$ and again, by Corollary 13, these points are in $N(Y, \mathcal{R}^2_{\geqq}, t)$. Also note that $y^3, y^5 \in N(Y, \mathcal{R}^2_{\geqq}, t)$ and $y^6 \notin N(Y, \mathcal{R}^2_{\geqq}, t)$ because $t(y^4) \leq_{\mathcal{R}^2_{\geqq}} y^6$. Thus $N(Y, \mathcal{R}^2_{\geqq}, t) = \{y^1, y^2, y^3, y^4, y^5\}$.*

Proposition 2.6.3. *Let Y be a set in $\mathcal{R}^p$, C be a convex cone in $\mathcal{R}^p$ and $t_1, t_2 : \mathcal{R}^p \mapsto \mathcal{R}^p$ be two tolerance functions such that $t_1(y) \leqq_C t_2(y)$ for all $y \in Y$. Then $N(Y, C, t_1) \subseteq N(Y, C, t_2)$.*

Proof. Since $t_1(y) \leqq_C t_2(y)$, we have $t_2(y) - t_1(y) \in C$, or equivalently, $t_2(y) \in t_1(y) + C$. This implies that $t_2(y) + C \subseteq t_1(y) + C + C + \subseteq t_1(y) + C$ since C is a convex cone. Since y is arbitrary, for all $y \in Y$ we obtain $t_2(y) + C \subseteq t_1(y) + C$.

Let $y' \in N(Y, C, t_1)$. By Definition 2.6.4, there does not exist $y \in Y$ such that $t_1(y) \leq_C y'$, or equivalently, $y' \in t_1(y) + C$. This implies that there does not exist $y \in Y$ such that $y' \in t_2(y) + C$. Thus $y' \in N(Y, C, t_2)$. Since y' is an arbitrary element of Y, we have $N(Y, C, t_1) \subseteq N(Y, C, t_2)$. $\square$

The final result of this section refers to the special case of Y being a subset of $\mathcal{Z}^p_>$ and the tolerance function $t(y) = (1+\epsilon)y$. Proposition 2.6.4 reveals that the set of all t-nondominated points reduces to the Pareto set for a certain magnitude of the ϵ. The result is significant for discrete or combinatorial MOPs since this tolerance function is typically used for that class of problems.

Let $||y||_{\mathtt{p}}$ denote the $\mathtt{p}$ norm of $y \in Y$ for $\mathtt{p} \in [1, \infty)$.

Proposition 2.6.4. *Let Y be a set in $\mathcal{Z}^p_>$ and $t : \mathcal{R}^p \mapsto \mathcal{R}^p$ be a tolerance function such that $t(y) = (1+\epsilon)y$ for $y \in Y$, where $0 \leq \epsilon \leq \frac{1}{\max\limits_{y \in Y}\{||y||_{\mathtt{p}}\}}$. Then $N(Y, \mathcal{R}^p_{\geqq}, t) = N(Y, \mathcal{R}^p_{\geqq})$.*

Proof. By Corollary 13, $N(Y, \mathcal{R}^p_{\geqq}) \subseteq N(Y, \mathcal{R}^p_{\geqq}, t)$. We show that $N(Y, \mathcal{R}^p_{\geqq}, t) \subseteq N(Y, \mathcal{R}^p_{\geqq})$. Let $y' \in N(Y, \mathcal{R}^p_{\geqq}, t)$. By Definition 2.6.4, there does not exist $y \in Y$ such that $t(y) \leq_{\mathcal{R}^p_{\geqq}} y'$, or equivalently, there does not exist $d \in \mathcal{R}^p_{\geqq}, d \neq 0$, such that $y' = t(y) + d$. That is,

$$\text{there does not exist } y \in Y \text{ and } d \in \mathcal{R}^p_{\geqq},\ d \neq 0 \text{ such that } y' = y + \epsilon y + d. \qquad (2.3)$$

If $\epsilon = 0$, (2.3) implies that $y' \in N(Y, \mathcal{R}^p_{\geqq})$.

Assume now $0 < \epsilon \leq \frac{1}{\max\{||y||_{\mathtt{p}}\}}$ for $y \in Y$. Then for every $y \in Y$ we have $0 < \epsilon y_i < 1$ for $i = 1, 2, \ldots, p$. Thus, $y_i < y_i + \epsilon y_i < y_i + 1$ for $i = 1, 2 \ldots, p$, or, equivalently, for any $y \in Y$

$$y < y + \epsilon y < y + (1, 1, \ldots, 1)^T. \qquad (2.4)$$

Since $Y \subset \mathcal{Z}^p_>$, by (2.4), there does not exist $\hat{y} \in Y$ such that $\hat{y} = y + \epsilon y$. Then by (2.3), there does not exist $\hat{y} \in Y$ and $d \in \mathcal{R}^p_{\geqq}$, $d \neq 0$, such that $y' = \hat{y} + d$. That is, by Definition 2.2.3, $y' \in N(Y, \mathcal{R}^p_{\geqq})$, which proves the result. $\square$

Example 2.6.3. *Let Y be the set as defined in Example 2.2.4. We have $N(Y, \mathcal{R}^2_{\geqq}) = \{y^1, y^2, y^4\}$. Let $t : \mathcal{R}^2 \mapsto \mathcal{R}^2$ such that $t(y) = (1+\epsilon)y$ where $\epsilon = 0.1 < \frac{1}{\max\limits_{y \in Y}\{||y||_2\}} = \frac{1}{\sqrt{9^2+3^2}}$. Note that $N(Y, \mathcal{R}^2_{\geqq}, t) = \{y^1, y^2, y^4\} = N(Y, \mathcal{R}^2_{\geqq})$.*

Remark 2.6.2. *The upper bound of the ϵ given in Proposition 2.6.4 can be found as the inverse of the optimal value of the single objective optimization problem $\max\limits_{y \in Y} ||y||_{\mathtt{p}}$ whose complexity depends on the norm selected. Naturally, the ℓ_1 norm yields a problem of lowest complexity. Furthermore, if the ℓ_∞-norm is used, then Proposition 2.6.4 holds for $0 \leq \epsilon < \frac{1}{\max\{||y||_{\ell_\infty}\}}$.*

In this paper we have proposed a unified approach to representing solution sets in multiobjective optimization. We have defined covers and approximations, and collected and proved their properties. The approach is tolerance-based and is relevant to a number of approximating algorithms in the literature. The introduced tolerance function leads to t-dominance which generalizes the concept of ϵ-nondominated points.

Further research directions are motivated by this paper. Based on the results of this section, algorithms for computing covers or approximation sets for complex multiobjective decision making problems may be designed. Additional properties of covers and approximations can also be studied in the context of the specific MOP they refer to.

Chapter 3

Pareto Set Approximation for the Multiobjective Set Covering Problem

3.1 Introduction

Multiobjective combinatorial optimization (MOCO) problems involve optimizing more than one objective function on a finite set of feasible solutions. Some well-known MOCO problems include the traveling salesman problem (TSP), the set covering problem (SCP), the minimum spanning tree problem (MSTP), and the knapsack problem (KP). During the past decades the interest in solving MOCO problems has been growing and surveys summarizing those efforts are given by Ehrgott ([24]), Ehrgott and Gandibleux ([26]), and Ulungu and Teghem ([100]). Because there may not exist a single optimal solution to a MOCO problem as the objective functions are in conflict with each other, a solution set is defined based on the Pareto concept of optimality. Solving a MOCO problem is then understood as computing the elements in the Pareto set. In this paper attention is given to the multiobjective set covering problem (MOSCP), a challenging MOCO problem that has not been much studied.

The MOSCP has the same structure as the well-known single objective set covering problem (SOSCP). An instance of the SCP consists of a finite set of the items and a family of subsets of the items so that every item belongs to at least one of the subsets in the family. When we consider the SOSCP, each set in the family has a positive scalar cost. The goal of the SOSCP is to determine a subset of sets, among the sets in the family, so that all items are included by at least one set in the subset and the total costs of the selected sets is minimized. When there are p scalar costs for each set in the family, the SCP is called the MOSCP.

The SCP is in the category of NP problems and it is shown to be NP-complete by Richard ([89]). Therefore, the SOSCP and MOSCP are also NP-hard problems. For NP-hard problems, we are typically interested in finding a near optimal solution, that is, a solution that yields the objective value that is worse than the optimal objective value by a factor of $\rho > 0$. An algorithm providing a near optimal solution with a factor ρ is called a ρ-approximation algorithm. Chvátal ([15]) and Vazirani ([102]) propose polynomial-time approximation algorithms for the SOSCP. Chvátal's ([15]) algorithm has the factor ρ being a function of the cardinality of the largest subset while Vazirani's ([102]) algorithm has the factor equal to $\log m$ where m is the number of items. The SOSCP is a well-studied problem and different methods have been proposed in the literature to address it ([13], [70]).

The MOSCP has not received as much attention as the SOSCP and only a few studies are found in the literature. Some real-world application problems such as the emergency medical service problem ([20]), the reserve site selection problem ([68]) are modelled as the bi-objective SCP (BOSCP). Liu ([61]) proposes a heuristic algorithm generating only one solution of the MOSCP. Saxena and Arora ([96]) formulate the SCP with quadratic objective functions and develop a heuristic enumeration technique for solving the MOSCP. The authors convert the quadratic objective functions to linear objective functions by assuming that all objective functions are differentiable and use the Gomory cut technique to get the efficient solutions. Jaszkiewicz ([53], [52]) provides a comparative study of multiobjective metaheuristics for the BOSCP. In particular, nine well-known multiobjective metaheuristics are compared with a new algorithm called the Pareto memetic algorithm (PMA). The performance of the multiobjective metaheuristics for the BOSCP depends on the problem structure. Prins and Prodhon ([85]) propose a heuristic-based two-phase method to find the Pareto set of the BOSCP. In the first phase, the scalarized SCP is solved with a heuristic to generate a subset of the Pareto set called the supported Pareto set. In the second phase, a heuristic algorithm searches for the Pareto points located between two supported Pareto points. This heuristic optimizes one objective function at a time and requires that this SOSCP be reformulated by Lagrangian relaxation. Lust et al. ([64]) adapt a very large-scale neighborhood search ([3]) for the MOSCP and compare average running times of the adaptation with the PMA and the TPM for the BOSCP. The performance of their algorithm also depends on the problem structure.

All the studies reviewed above propose heuristic approaches to obtaining the Pareto set of the MOSCP. As the MOSCP is an NP-hard problem, from the computational point of view, approximating the Pareto set is the right direction of research. However, neither the authors quoted above claim that their methods approximate the entire Pareto set nor they provide performance guarantee for their algorithms. Contrary to those approaches, the objective of this paper is to propose algorithms for approximating the Pareto set of the MOSCP and provide the information about the approximation that the existing algorithms lack.

To accomplish this, we follow the definition of Pareto set approximation by Papadimitriou and Yannakakis in 2000 ([81]), who recognize that the Pareto set of a MOCO problem is typically exponential in its size and therefore, finding all Pareto points is computationally infeasible. Even for two objective functions, determining whether a point belongs to the Pareto set is an NP problem. They propose an approximation of the Pareto set which they call the $(1+\epsilon)$-approximate Pareto set and define the approximation as a set of solutions such that for every Pareto point there exists another point within a factor of $(1+\epsilon)$ where $\epsilon > 0$. This definition has already been considered for other MOCO NP-hard problems such as the TSP ([4]) and the KP ([30]), but not yet for the MOSCP.

This paper proposes two methods to approximate the Pareto set of the MOSCP. The first algorithm is an application of the greedy algorithm for the SOSCP by Vazirani ([102]) to the weighted-sum formulation of the MOSCP. The second algorithm, although also motivated by the greedy algorithm, solves the MOSCP in the vector form without scalarization. Both algorithms appear to be first approaches in the literature to approximate the Pareto set of the MOSCP with known factors. The first algorithm obtains a set of feasible solutions of the MOSCP approximating the supported Pareto set of the MOSCP while the second algorithm obtains a set of feasible solutions of the MOSCP approximating the weak Pareto set of the MOSCP, a feature not available for the existing methods.

The paper is organized as follows. Section 3.2 provides the formulation of the MOSCP, terminology, and discusses two general approaches (exact and approximate) to computing the Pareto points of the MOSCP. In Section 3.3, we present the two algorithms and derive their approximation factors. In Section 3.4, we show computational results on the BOSCPs and propose experimental measures of the quality of the computed approximations. We then compare the experimental and theoretical results. The paper is concluded in Section 3.5.

3.2 Problem formulation

In the MOSCP, there is a set of m items, $E = \{e_1, e_2, \ldots, e_m\}$ with the index set $I = \{i : i = 1, 2, \ldots, m\}$, and a set of n subsets of E, $S = \{S_1, S_2, \ldots, S_n\}$ with the index set of $J = \{j \ : \ j = 1, 2, \ldots, n\}$. The items are grouped into subsets of E and an item e_i in E is covered by a set S_j provided e_i in S_j. An instance of the SCP is given by the sets E and S. The binary coefficient a_{ij} is equal to 1 if an item e_i is covered by a set S_j and otherwise a_{ij} is equal to 0 for $i \in I$ and $j \in J$. A cover is defined as a sub-collection $\{S_j : \jmath \in J^* \subseteq J\}$ which is a subset of S such that all items of E are covered, where J^* is the index set of selected sets for the sub-collection. Let $x \in \mathcal{R}^n$ be the decision variable defined as follows,

$$x_j = \begin{cases} 1 & \text{if } S_j \text{ is selected for a cover} \\ 0 & \text{otherwise,} \end{cases} \quad \text{for } j = 1, 2, \ldots, n.$$

As mentioned in the Introduction, in the SCP each item must be covered by at least one set, i.e., a cover is sought to cover all items. Thus the feasible region X is defined as
$X = \{x \in \mathcal{R}^n : \sum\limits_{j \in J} a_{ij} x_j \geq 1$ for $i = 1, 2, \ldots, m$ and $x_j \in \{0, 1\}$ for $j = 1, 2, \ldots, n\}$.
Every feasible vector $x \in X$ is associated with a cover and vice versa.

The MOSCP has p conflicting objectives and let $c_j^q > 0$ be the cost of a set S_j with respect to objective q for $q = 1, 2, \ldots, p$. The cost of a cover with respect to objective q is given by $\sum\limits_{j \in J^*} c_j^q$. In the MOSCP the goal is to find a cover such that the costs with respect to all objective functions are minimized.
The MOSCP can be presented as follows:

$$\begin{aligned} \min\ z(x) = \ & \left[z_1(x) = \sum_{j=1}^{n} c_j^1 x_j, \quad z_2(x) = \sum_{j=1}^{n} c_j^2 x_j, \quad \ldots, \quad z_p(x) = \sum_{j=1}^{n} c_j^p x_j \right] \\ & \text{subject to} \quad x \in X. \end{aligned} \tag{3.1}$$

3.2.1 Preliminaries and basic definitions

Let $\mathcal{R}^p$ be a finite dimensional Euclidean vector space. We first introduce some basic notations. For $y^1, y^2 \in \mathcal{R}^p$, to define an ordering relation on $\mathcal{R}^p$, the following notation will be used.

1. $y^1 \leqq y^2$ if $y_k^1 \leq y_k^2$ for all $k = 1, 2, \ldots, p$

2. $y^1 \leq y^2$ if $y_k^1 \leq y_k^2$ for all $k = 1, 2, \ldots, p$ and $y^1 \neq y^2$

3. $y^1 < y^2$ if $y_k^1 < y_k^2$ for all $k = 1, 2, \ldots, p$

In particular, using componentwise orders, the nonzero orthant of $\mathcal{R}^p$ is defined as $\mathcal{R}^p_{\geq} = \{y \in \mathcal{R}^p : \ y \geq 0\}$ and positive orthant of $\mathcal{R}^p$ is defined as $\mathcal{R}^p_{>} = \{y \in \mathcal{R}^p : \ y > 0\}$. Solving the MOSCP is understood as finding its efficient solutions and Pareto outcomes.

Definition 3.2.1. *A point $x^* \in X$ is called*

1. *a weakly efficient solution of the MOSCP if there does not exist $x \in X$ such that $z(x) < z(x^*)$.*
2. *an efficient solution of the MOSCP if there does not exist $x \in X$ such that $z(x) \leq z(x^*)$.*

The set of all efficient solutions and the set of all weakly efficient solutions are denoted by X_E and X_{wE} respectively. The set of all attainable outcomes, Y, for feasible solutions, $x \in X$, is obtained by evaluating the p objective functions. That is $Y := z(X) \subset \mathcal{R}^p$. The image $z(x) \in Y$ of a (weakly) efficient solution is called a (weak) Pareto outcome. The image of (X_{wE}) X_E is denoted by $(P_w(Y))$ $P(Y)$ and is referred to as the (weak) Pareto set. Given the definition of an (weakly) efficient solution of the MOSCP, we define a (weak) Pareto cover for the MOSCP.

Definition 3.2.2. *A (weak) Pareto cover is a cover that is associated with an (weakly) efficient solution of the MOSCP.*

3.2.2 Finding efficient solutions of the MOSCP

The approximation algorithms we propose in this paper are developed based on two exact methods for finding the efficient solutions of multiobjecive optimization problems (MOOPs). These methods are the weighted-sum method and the weighted max-ordering method. In this section, we first briefly review these two methods and include the results needed when proving the accuracy of the algorithms. We then present the concepts of approximation as another approach to solving MOOPs.

3.2.2.1 Exact methods

The weighted-sum method is a well-known scalarization method used for MOOPs to find supported efficient solutions. The idea of this method is to convert the MOOP into a singleobjective optimization problem (SOOP) using a convex combination of objectives. In this method each criterion q is assigned a weighting coefficient $\lambda_q \geq 0$ and the SOOP is solved over the same feasible region. The weighted-sum problem associated with the MOSCP can be written as follows:

$$\begin{aligned} &\min \sum_{q=1}^{p} \lambda_q z_q(x) \\ &\text{subject to } x \in X \end{aligned} \tag{3.2}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_p) \in \mathcal{R}^p_{\geq}$.

Definition 3.2.3. *Let $x \in X_E$. If there is some $\lambda \in \mathcal{R}^p_{>}$ such that $x \in X_E$ is an optimal solution of problem (3.2), then x is called a supported efficient solution and the image $z(x) \in Y$ is called a supported Pareto outcome of the MOSCP.*

The set of all supported efficient solutions of the MOSCP is denoted by X_{sE}. The image of X_{sE} is denoted by $P_s(Y)$ and is referred to as the supported Pareto set.

Definition 3.2.4. *A supported Pareto cover is a cover that is associated with a supported efficient solution of the MOSCP.*

The following result is readily available for the MOSCP ([29]).

Proposition 3.2.1. *Let $x^* \in X_E$. Then $x^* \in X_{sE}$ of the MOSCP if and only if there exists $\lambda \in \mathcal{R}^p_{>}$ such that x^* is an optimal solution of problem (3.2).*

The underlying concept of the weighted max-ordering method is to minimize the highest (worst) objective function value, z_q. The weighted max-ordering problem associated with the MOSCP can be written as follows:

$$\begin{aligned} &\min \max_{q=1,2,\ldots,p} \lambda_q z_q(x) \\ &\text{subject to } x \in X \end{aligned} \tag{3.3}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_p) \in \mathcal{R}^p_{\geq}$.

The following results is useful for the MOSCP ([29]).

Proposition 3.2.2. *Let $x^* \in X_E$. Then $x^* \in X_{wE}$ of the MOSCP if and only if there exists $\lambda \in \mathcal{R}^p_>$ such that x^* is an optimal solution of problem (3.3).*

These two methods imply that there are different ways of comparing vectors. The comparison of weighted aggregation of vectors can be used to define a preference relation on $\mathcal{R}^p$ as given in Definition 3.2.5. We write $\underset{ws}{\succeq}$ to denote the preference relation with respect to the weighted aggregation of vectors. This relation is called the weighted-sum preference relation.

Definition 3.2.5. *Let $y^1, y^2 \in \mathcal{R}^p$ and $\lambda \in \mathcal{R}^p_{\geq}$. Vector y^1 is preferred to vector y^2 with respect to the weighted-sum preference relation, denoted as $y^1 \underset{ws}{\succeq} y^2$, if and only if $\sum_{q=1}^{p} \lambda_q y^1_q \leq \sum_{q=1}^{p} \lambda_q y^2_q$.*

The ordering concept of problem (3.3), can also be used to compare vectors. In this case, we compare maximum components of vectors to define a preference relation on $\mathcal{R}^p$ as given in Definition 3.2.6. We write $\underset{mo}{\succeq}$ to denote the resulting preference relation and refer to it as the max-ordering preference relation.

Definition 3.2.6. *Let $y^1, y^2 \in \mathcal{R}^p$ and $\lambda \in \mathcal{R}^p_{\geq}$. Vector y^1 is preferred to vector y^2 with respect to the max-ordering preference relation, denoted as $y^1 \underset{mo}{\succeq} y^2$, if and only if $\max_{q=1,2,\ldots,p} \lambda_q y^1_q \leq \max_{q=1,2,\ldots,p} \lambda_q y^2_q$.*

The preference relations given in Definition 3.2.5 and Definition 3.2.6 are used in Section 3 for the development of the algorithms.

3.2.2.2 Approximation

As mentioned in the Introduction, approximating the Pareto set with a performance guarantee is a motivating challenge. Following the definition of Papadimitriou and Yannakakis in 2000 ([81]), we define the $(1+\epsilon)$-approximate Pareto set for the MOSCP.

Definition 3.2.7. *Let $\epsilon \in \mathcal{R}$ be a positive scalar. The $(1+\epsilon)$-approximate Pareto set, P^{ϵ}, of the MOSCP is a set of outcomes in Y such that for every $y^* \in P(Y)$, there exists an outcome $\bar{y}$ in P^{ϵ} such that $\bar{y} \leqq (1+\epsilon)y^*$.*

When the weak Pareto set, $P_w(Y)$, is approximated, P_w^{ϵ} is called $(1+\epsilon)$-approximate weak Pareto set. Definition 3.2.7 can be modified when the approximation of the supported Pareto set, $P_s(Y)$, is of interest.

Definition 3.2.8. *Let $\epsilon \in \mathcal{R}$ be a positive scalar. The $(1+\epsilon)$-approximate supported Pareto set, P_s^{ϵ}, of the MOSCP is a set of outcomes in Y such that for every $y^* \in P_s(Y)$, there exist an outcome $\bar{y}$ in P_s^{ϵ} and a weight $\lambda \in \mathcal{R}^p_{\geq}$ such that $\sum_{q=1}^{p} \lambda_q \bar{y}_q \leq (1+\epsilon) \sum_{q=1}^{p} \lambda_q y_q^*$.*

3.3 Approximating the Pareto set of the MOSCP

In this section, we present two deterministic polynomial-time algorithms to approximate the Pareto set of the MOSCP. The first algorithm computes a $(1+\epsilon)$-approximate supported Pareto set of the MOSCP while the other algorithm computes a $(1+\epsilon)$-approximate weak Pareto set of the MOSCP.

3.3.1 Scalar and vector cost effectiveness

We first present the concept of the cost effectiveness of a set. The key idea of our approach is based on the following observation: when selecting a set to be in a minimum cost cover, we need to consider not only the cost of the set but also the coverage of the set, that is, the items this set covers. For example, let $S_1 = \{e_1, e_2, e_3, e_4\}$ and $S_2 = \{e_5, e_6\}$ be two sets with $c_1^1 = c_2^1 = 1$. Intuitively, it is clear that selecting S_1 is more beneficial than selecting S_2 as it contains more items even though both sets have the same costs. Therefore, when we select a minimum cost cover, it seems reasonable to choose a set having a small cost and a large coverage. Consequently, in the construction of the algorithms, when selecting a set we consider these two aspects: the minimum cost and the maximum coverage. This is equivalent to selecting a set having a small ratio of cost to coverage. The ratio (cost/coverage) is called the cost effectiveness of the set and is denoted by α_j for a set S_j. We propose two concepts for defining cost effectiveness. The first concept is based on the weighted-sum method and defines the cost effectiveness as a scalar while the second concept is based on the max-ordering method and defines the cost effectiveness as a vector. To keep derivations simple, we use the same notation α to denote cost effectiveness based on both approaches; however, it will be clear from the context whether we use the scalar or vector cost effectiveness.

Based on the weighted-sum method, the scalar cost effectiveness of a set S_j is defined as follows:

$$\alpha_j = \sum_{q=1}^{p} \lambda_q c_j^q / |S_j|, \tag{3.4}$$

where $|S_j|$ is the cardinality of the set S_j and $\lambda \in \mathcal{R}^p_{\geq}$.

Then to select a set to include in a cover, we use Definition 3.3.1.

Definition 3.3.1. *A set S_{j_1} is preferred to a set S_{j_2}, $j_1 \neq j_2$, with respect to the weighted-sum preference relation if $\alpha_{j_1} \underset{ws}{\succeq} \alpha_{j_2}$.*

For example, assume that the sets S_1 and S_2 have two costs. Let $c_1^1 = 1,\ c_1^2 = 4$ and let $c_2^1 = 1,\ c_2^2 = 1$. Let $\lambda = 1/3$. The cost effectiveness of S_1 is $\alpha_1 = 9/12$ and that of S_2 is $\alpha_2 = 4/6$. Thus α_2 is preferred to α_1 by Definition 3.3.1. Therefore selecting S_2 is better than selecting S_1.

Based on the weighted max-ordering method, the cost effectiveness of a set is a vector of cost effectiveness ratios, where each ratio is the cost effectiveness with respect to one objective. The vector cost effectiveness of a set S_j is defined as follows:

$$\alpha_j = \left[\lambda_1 c_j^1/|S_j|, \lambda_2 c_j^2/|S_j|, \ldots, \lambda_p c_j^p/|S_j|\right]^T, \tag{3.5}$$

where $|S_j|$ is the cardinality of the set S_j and $\lambda \in \mathcal{R}^p_{\geq}$.

We use Definition 3.3.2 to select a set to include in a cover with respect to the max-ordering preference relation.

Definition 3.3.2. *A set S_{j_1} is preferred to a set S_{j_2}, $j_1 \neq j_2$, with respect to the max-ordering preference relation if $\alpha_{j_1} \underset{mo}{\succeq} \alpha_{j_2}$.*

Considering the same example in the vector form, the cost effectiveness vector of S_1 is $\alpha_1 = [\ 1/12,\ 8/12\ ]^T$ and that of S_2 is $\alpha_2 = [\ 1/6,\ 2/6\ ]^T$. In this case, we have that α_2 is preferred to α_1 by Definition 3.3.2. Thus selecting S_2 is better than selecting S_1.

3.3.2 Algorithms

The concepts of scalar and vector cost effectiveness lead to the development two approximation algorithms. A generic algorithm using either of the two concepts and returning a cover is first presented. Two procedures are used in this generic algorithm. Procedure 1 is constructed based on the weighted-sum approach, and it returns the preferred set S_{j^*} and the scalar cost effectiveness α_{j^*} according to Definition 3.3.1, while Procedure 2 is constructed based on the max-ordering approach, and it returns the preferred set S_{j^*} and the vector cost effectiveness α_{j^*} according to Definition 3.3.2. The pseudo-codes of Generic Algorithm and two procedures are given in Algorithm 1, Procedure 1 and Procedure 2, respectively. The generic algorithm with Procedure 1 is called Algorithm 1 and that with Procedure 2 is called Algorithm 2. In the generic algorithm, the symbols $\bar{E}$, $\bar{J}$ and C denote the set of currently covered items in E, the index set of selected sets for covering items in E, and a cover, respectively. We assign α_{j^*} to the items covered by the set S_{j^*} as their prices and refer to α_{j^*} as the price of an item covered by S_{j^*}. We let $p(e_i)$ denote the price of item e_i. Based on Definitions 3.3.1 and 3.3.2, $p(e_i)$ may be a scalar or vector price.

3.3.2.1 Performance of Algorithm 1

The structure of Algorithm 1 is similar to the greedy algorithm ([102]) for the SOSCP. However; we have adapted it to approximate the set of supported Pareto outcomes of the MOSCP according to Definition 3.2.8. In other words, the algorithm approximately solves a collection of SOSCPs obtained from scalarizing the MOSCP with weights $\lambda \in \mathcal{R}^p_>$. The outline of Algorithm 1 is the following: the algorithm starts with the empty sets $\bar{E}$, $\bar{J}$

Algorithm 1 Generic Algorithm for the MOSCP

```
 1: Input: $E$, $S$, $c^q$ for $q = 1, 2, \dots, p$, $\lambda \in \mathcal{R}^p_>$ where $\sum_{q=1}^{p} \lambda_q = 1$.
 2: Initialization: $\overline{E} = \emptyset, \overline{J} = \emptyset$ and $C = \emptyset$.
 3: while $\overline{E} \neq E$ do
 4:     Call Procedure 1 or 2 to obtain $\alpha_{j^*}$ and $S_{j^*}$.
 5:     for $i = 1 \to n$ do
 6:         if $e_i \in S_{j^*} \cap (E \setminus \overline{E})$ then
 7:             $p(e_i) = \alpha_{j^*}$, $\overline{J} = \overline{J} \cup j^*$, $\overline{J} \subseteq J$
 8:         end if
 9:     end for
10: end while
11: Return: Cover, $C = \{S_j : j \in \overline{J}\}$.
```

Pseudo-code of the generic algorithm

and the empty set as the current cover, that is, initially $C = \emptyset$. In the main step, Procedure 1 is called to calculate the scalar cost effectiveness values, $\alpha_j s$, for all sets based on the scalarized costs and uncovered items, and to determine a set to be added to the current cover. Once the best set, S_{j^*}, has been selected, the index j^* is added to the index set, $\bar{J}$, and all items in S_{j^*} are added to $\bar{E}$. For each item e_i covered in this step, the price is set as $p(e_i) = \alpha_{j^*}$. Algorithm 1 is run until all items in E have been covered. Upon termination, the algorithm yields a cover associated with a feasible solution, $\bar{x}$, where the components of $\bar{x}$ identify the sets selected to be in the cover.

To evaluate the performance of Algorithm 1, we show that $\bar{x}$, the solution yielded by Algorithm 1 with a weight λ^*, can be used to get a bound on the optimal objective function value of problem (3.2) obtained for the same weight λ^*. The key observation about Algorithm 1 is given in Lemma 3.3.1 in which we estimate the price $p(e_k)$ of an item e_k assigned by Algorithm 1, where e_k is an item covered in the iteration k.

Lemma 3.3.1. *Let $x^* \in X$ be an optimal solution of problem (3.2) associated with $\lambda^* \in \mathcal{R}^p_>$. Let e_k be an item covered in the k^{th} iteration of Algorithm 1. Then*

$$p(e_k) \leq \sum_{q=1}^{p} \lambda^*_q z_q(x^*)/(m - k - 1).$$

Procedure 1 Weighted-sum method

```
1: Initialization:        $\alpha_{j^*} = \infty$

2: for $j = 1 \to n$ do
3:     if  $j \in J \setminus \overline{J}$ then
4:         $\alpha_j = \sum_{q=1}^{p} \lambda_q \alpha_j^q = \frac{\sum_{q=1}^{p} \lambda_q c_j^q}{|S_j \cap (E \setminus \overline{E})|}$
5:         if  $\alpha_j \le \alpha_{j^*}$ then
6:             $\alpha_{j^*} = \alpha_j$
7:         end if
8:     end if
9: end for

10: Return: $\alpha_{j^*}$ and $S_{j^*}$
```

Pseudo-code of Procedure 1

Proof. By Proposition 3.2.1, $x^* \in X_{sE}$. Let S^* be the supported Pareto cover associated with x^*. Then $S^* = \{S_1^*, S_2^*, \ldots, S_{r^*}^*\}$ where $\{S_1^*, S_2^*, \ldots, S_{r^*}^*\}$ is a collection of sets from the family S and r^* is the number of sets in the supported Pareto cover. We know that $E = \bigcup_{l=1}^{r^*} S_l^*$ and $\bigcup_{l=1}^{r*} S_l^*$ is a supported Pareto cover with the cost $\sum_{l=1}^{r^*} \lambda_l^* c^l$. At any iteration of Algorithm 1, the uncovered items are given by $E \setminus \overline{E}$. Since a supported Pareto cover covers all items, at any iteration, $E \setminus \overline{E}$ can be expressed as $E \setminus \overline{E} = \bigcup_{l=1}^{r^*} (S_l^* \cap (E \setminus \overline{E}))$. Thus we have,

$$|E \setminus \overline{E}| \le \sum_{l=1}^{r^*} |S_l^* \cap (E \setminus \overline{E})|. \tag{3.6}$$

Note that at any iteration, the uncovered items in S_l^* are contained in $S_l^* \cap (E \setminus \overline{E})$ for $l = 1, 2, \ldots, r^*$ and if all items in S_l^* are covered then $S_l^* \cap (E \setminus \overline{E}) = \emptyset$. Consider the iteration in which an item e_k is covered. Let S_{j^*} be the set selected in this iteration to cover item e_k and let α_{j^*} be the scalar cost effectiveness of this set. We obtain a bound for α_{j^*} using the supported Pareto cover S^*. Let $S_l^* \in S^*$ and $\alpha_l = \sum_{q=1}^{p} \lambda_q^* \alpha_l^q$ be the scalar cost effectiveness of S_l^*. We consider the following two cases.

Case 1: Let $l \in J \setminus \overline{J}$, that is, some items in S_l^* are not covered by Algorithm 1 and thus the set S_l^* is a candidate for S_{j^*}. In this case, the uncovered items in S_l^* are given by $S_l^* \cap (E \setminus \overline{E})$ and

$$S_l^* \cap (E \setminus \overline{E}) = S_j \cap (E \setminus \overline{E}) \text{ for some } j \in J \setminus \overline{J}.$$

We calculate α_j values for all unselected sets based on formula (3.4) and select a best set using Definition 3.4. We obtain

$$\alpha_{j^*} = \sum_{q=1}^{p} \lambda_q^* \alpha_{j^*}^q \leq \sum_{q=1}^{p} \lambda_q^* \frac{c_l^q}{|S_l^* \cap (E \setminus \overline{E})|} \text{ for all } l \notin J \setminus \overline{J}.$$

That is, by Definition 3.2.5,

$$\alpha_{j^*} \underset{ws}{\succeq} \alpha_l \text{ for all } l \notin J \setminus \overline{J}. \tag{3.7}$$

Case 2: Let $l \notin J \setminus \overline{J}$, that is, all items in S_l^* are covered by Algorithm 1. In this case, $S_l^* \cap (E \setminus \overline{E}) = \emptyset$ and $\frac{\lambda_l^* c_l^q}{|S_l^* \cap (E \setminus \overline{E})|} = \infty$. Thus we obtain

$$\alpha_{j^*} = \sum_{q=1}^{p} \lambda_q^* \alpha_{j^*}^q \leq \sum_{q=1}^{p} \lambda_q^* \frac{c_l^q}{|S_l^* \cap (E \setminus \overline{E})|} \text{ for all } l \in J \setminus \overline{J}.$$

That is,

$$\alpha_{j^*} \underset{ws}{\succeq} \alpha_l \text{ for all } l \in J \setminus \overline{J}. \tag{3.8}$$

From (3.7) and (3.8) we conclude that

$$\alpha_{j^*} \underset{ws}{\succeq} \alpha_l \text{ for } l = 1, 2, \ldots, r^*.$$

Then, by Definition 3.3.1, we have

$$\alpha_{j^*} = \sum_{q=1}^{p} \lambda_q^* \alpha_{j^*}^q \leq \sum_{q=1}^{p} \lambda_q^* \frac{c_l^q}{|S_l^* \cap (E \setminus \overline{E})|} \text{ for } l = 1, 2, \ldots, r^*.$$

That is,

$$\alpha_{j^*}|S_l^* \cap (E \setminus \overline{E})| \leq \sum_{q=1}^{p} \lambda_q^* c_l^q \text{ for } l = 1, 2, \ldots, r^*. \tag{3.9}$$

By summing over all sets in the supported Pareto cover, inequality (3.9) becomes

$$\sum_{l=1}^{r^*}(\alpha_{j^*}|S_l^* \cap (E \setminus \overline{E})|) \leq \sum_{l=1}^{r^*}(\sum_{q=1}^{p} \lambda_q^* c_l^q). \tag{3.10}$$

Using inequality (3.6), we write inequality (3.10) as follows:

$$\alpha_{j^*}|(E \setminus \overline{E})| \leq \sum_{l=1}^{r^*}(\sum_{q=1}^{p} \lambda_q^* c_l^q). \tag{3.11}$$

Since $\sum_{l=1}^{r^*} \lambda_q^* c_l^q = \lambda_q^* z_q(x^*)$, inequality (3.11) can be rewritten as

$$\alpha_{j^*}|(E \setminus \overline{E})| \leq \sum_{l=1}^{r^*}(\sum_{q=1}^{p} \lambda_q^* c_l^q) = \sum_{q=1}^{p} \lambda_q^* z_q(x^*).$$

That is,

$$\alpha_{j^*} \leq \frac{\sum_{q=1}^{p} \lambda_q^* z_q(x^*)}{|(E \setminus \overline{E})|}. \tag{3.12}$$

In the iteration in which the item e_k is covered, the number of uncovered items is $(m-k+1)$. That is, $|(E \setminus \overline{E})| = m - k + 1$. Thus, inequality (3.12) gives the following:

$$\alpha_{j^*} \leq \frac{\sum_{q=1}^{p} \lambda_q^* z_q(x^*)}{m - k + 1}. \tag{3.13}$$

Since $p(e_k) = \alpha_{j^*}$, from inequality (3.13) we conclude $p(e_k) \leq \frac{\sum_{q=1}^{p} \lambda_q^* z_q(x^*)}{m-k+1}$. □

Corollary 14 shows that the objective value of problem (3.2) associated with the solution yield by Algorithm 1 is equal to the cost of covering all items.

Corollary 14. *If $\bar{x} \in X$ is a solution of the MOSCP yield by Algorithm 1 for $\lambda^* \in \mathcal{R}^p_>$, then $\sum_{q=1}^{p} \lambda^*_q z_q(\bar{x}) = \sum_{k=1}^{m} p(e_k)$.*

Proof. In Algorithm 1, the cost of each set selected in each iteration is distributed among the items covered in that iteration. Therefore, the cost of covering all items, at the termination of Algorithm 1, is equal to $\sum_{k=1}^{m} p(e_k)$. On the other hand, the cost of the selected cover is given by the objective value of problem (3.2), $\sum_{q=1}^{p} \lambda^*_q z_q(\bar{x})$. By definition, a cover covers all items and thus, we have,

$$\sum_{q=1}^{p} \lambda^*_q z_q(\bar{x}) = \sum_{k=1}^{m} p(e_k). \tag{3.14}$$

□

The solution yield by Algorithm 1 can be used to obtain a bound on the optimal objective function value of problem (3.2).

Theorem 3.3.1. *Let $\bar{x} \in X$ be a solution of the MOSCP yield by Algorithm 1 and x^* be an optimal solution of problem (3.2) associated with $\lambda^* \in \mathcal{R}^p_>$. Then*

$$\sum_{q=1}^{p} \lambda^*_q z_q(\bar{x}) \leq \log m \sum_{q=1}^{p} \lambda^*_q z_q(x^*). \tag{3.15}$$

Proof. Using Lemma 3.3.1, and summing over all items we get,

$$\sum_{k=1}^{m} p(e_k) \leq \sum_{k=1}^{m} \frac{\sum_{q=1}^{p} \lambda^*_q z_q(x^*)}{m-k+1} = (1 + 1/2 + \cdots + 1/m) \sum_{q=1}^{p} \lambda^*_q z_q(x^*).$$

As $(1 + 1/2 + \cdots + 1/m) \approx \log m$, we obtain

$$\sum_{k=1}^{m} p(e_k) \leq \log m \sum_{q=1}^{p} \lambda^*_q z_q(x^*). \tag{3.16}$$

By Corollary 14, inequality (3.16) gives inequality (3.15). □

Corollary 15 provides that the result of Theorem 3.3.1 can be used to construct an approximate supported Pareto set P_s^ϵ for the MOSCP.

Corollary 15. *For every $x \in X_{sE}$ of the MOSCP, there exist a weight vector $\lambda \in \mathcal{R}^p_>$ and a solution $\bar{x} \in X$ yield by Algorithm 1 satisfying the following condition:*

$$\sum_{q=1}^{p} \lambda_q z_q(\bar{x}) \leq \log m \sum_{q=1}^{p} \lambda_q z_q(x). \tag{3.17}$$

Proof. Let $x \in X_{sE}$. Then, by Proposition 3.2.1, there exists $\lambda \in \mathcal{R}^p_>$, such that x is an optimal solution of problem (3.2). We show that there exists a solution $\bar{x} \in X$ such that (3.17) holds. Suppose that Algorithm 1 is executed for λ and returns a cover associated with the solution $\bar{x}$. Then by Theorem 3.3.1, we have $\sum\limits_{q=1}^{p} \lambda_q z_q(\bar{x}) \leq \log m \sum\limits_{q=1}^{p} \lambda_q z_q(x)$, and (3.17) holds. □

According to Corollary 15, the bound on the optimal objective function value of problem (3.2) is $\log m$ for every $\lambda \in \mathcal{R}^p_>$. Using condition (3.17) and Definition 3.2.8, we obtain the following theoretical result.

If Algorithm 1 is run for all $\lambda \in \mathcal{R}^P_{\geq}$, for every solution in the set $P_s(Y)$ there is a solution in the set of solutions returned by Algorithm 1 satisfying condition (3.17). Thus, in view of Definition 3.2.8, the algorithm returns a $(1+\epsilon)$-approximate supported Pareto set of the MOSCP, where

$$1 + \epsilon = \log m \text{ and } m \text{ is the number of items.} \tag{3.18}$$

3.3.2.2 Performance of Algorithm 2

In this section, we present Algorithm 2 that approximately solves the MOSCP and is the first algorithm for approximating all weak Pareto outcomes of the MOSCP.

The outline of Algorithm 2 is the same as that of Algorithm 1 with the exception that Procedure 1 is replaced with Procedure 2 that calculates the vectors of cost effectiveness for all sets and determines a set to be added to the current cover.

Procedure 2 Max-ordering method

```
1: Initialization: $\alpha_{j^*} = (\infty, \infty, \dots, \infty)$

2: for $j = 1 \to n$ do
3:     if $j \in J \setminus \overline{J}$ then
4:         $\alpha_j = (\alpha_j^1, \alpha_j^2, \dots, \alpha_j^p)^T = \left(\lambda_1 \frac{c_j^1}{|S_j \cap (E \setminus \overline{E})|}, \lambda_2 \frac{c_j^2}{|S_j \cap (E \setminus \overline{E})|}, \dots, \lambda_p \frac{c_j^p}{|S_j \cap (E \setminus \overline{E})|}\right)^T$
5:         if $\max\{\alpha_j^1, \alpha_j^2, \dots, \alpha_j^p\} \leq \max\{\alpha_{j^*}^1, \alpha_{j^*}^2, \dots, \alpha_{j^*}^p\}$ then
6:             $\alpha_{j^*} = \alpha_j$
7:         end if
8:     end if
9: end for

10: Return: $\alpha_{j^*}$ and $S_{j^*}$
```

Pseudo-code of Procedure 2

To evaluate the performance of Algorithm 2, we show that the cost of the cover associated with $\bar{x}$, the solution yield by Algorithm 2 with a weight λ^*, can be used to get a bound on the cost of a weak Pareto cover associated with the optimal objective function value of problem (3.3) obtained for the same weight λ^*. The main observation about Algorithm 2 is given in Lemma 3.3.2 in which we estimate the price $p(e_k)$ of an item e_k assigned by Algorithm 2, where e_k is an item covered in the iteration k.

In the proofs presented in this section, we use the symbol $\lceil a \rceil$ to denote the ceiling of a which is the smallest integer not less than a, where a is a real number. Given the data of the MOCSP, we define

$$\begin{aligned} c_{\min}^{k_1} &= \min\{c_1^{k_1}, c_2^{k_1}, \dots, c_n^{k_1}\} \\ c_{\max}^{k_2} &= \max\{c_1^{k_2}, c_2^{k_2}, \dots, c_n^{k_2}\}, \end{aligned} \tag{3.19}$$

for $k_1, k_2 \in \{1, 2, \dots p\}$.

Lemma 3.3.2. *Let $x^* \in X$ be an optimal solution of problem (3.3) associated with λ^* for $\lambda^* \in \mathcal{R}^p_>$. Let e_k be an item covered in the k^{th} iteration of Algorithm 2. Then*

$$p(e_k) \leqq \delta \begin{pmatrix} z_1(x^*)/(m-k+1) \\ z_2(x^*)/(m-k+1) \\ \vdots \\ z_p(x^*)/(m-k+1) \end{pmatrix},$$

where $\delta = \max\limits_{\substack{k_1,k_2=1,2,\ldots,p, \\ k_1 \neq k_2}} \{\lambda^*_{k_2} \lceil \frac{c^{k_2}_{\max}}{c^{k_1}_{\min}} \rceil\}$.

Proof. By Proposition 3.2.2, $x^* \in X_{wE}$. Let S^* be the weak Pareto cover associated with x^*. Then $S^* = \{S^*_1, S^*_2, \ldots, S^*_{r^*}\}$ where $\{S^*_1, S^*_2, \ldots, S^*_{r^*}\}$ is a collection of sets from the family S and r^* is the number of sets in the weak Pareto cover. We know that $E = \bigcup\limits_{l=1}^{r^*} S^*_l$ and $\bigcup\limits_{l=1}^{r^*} S^*_l$ is a weak Pareto cover with the cost $[\lambda^*_1 \sum\limits_{l=1}^{r^*} c^1_l, \ldots, \lambda^*_p \sum\limits_{l=1}^{r^*} c^p_l]^T$. At any iteration of Algorithm 2, the uncovered items are given by $E \setminus \overline{E}$. Since a weak Pareto cover covers all items, at any iteration, $E \setminus \overline{E}$ can be expressed as $E \setminus \overline{E} = \bigcup\limits_{l=1}^{r^*} (S^*_l \cap (E \setminus \overline{E}))$. Thus we have,

$$|E \setminus \overline{E}| \leq \sum_{l=1}^{r^*} |S^*_l \cap (E \setminus \overline{E})|. \tag{3.20}$$

Note that at any iteration, the uncovered items in S^*_l are contained in $S^*_l \cap (E \setminus \overline{E})$ for $l = 1, 2, \ldots, r^*$ and if all items in S^*_l are covered then $S^*_l \cap (E \setminus \overline{E}) = \emptyset$. Consider now the iteration in which an item e_k is covered. Let S_{j^*} be the set selected in this iteration to cover the item e_k and let α_{j^*} be the vector cost effectiveness of this set. We obtain a bound for α_{j^*} using the weak Pareto cover S^*. Let $S^*_l \in S^*$ and $\alpha_l = [\alpha^1_l, \ldots, \alpha^p_l]^T$ be the cost effectiveness of S^*_l. We consider the following two cases.

Case 1: Let $l \in J \setminus \overline{J}$, that is, some items in S^*_l are not covered by Algorithm 2 and thus the set S^*_l is a candidate for S_{j^*}. In this case, the uncovered items in S^*_l are given by $S^*_l \cap (E \setminus \overline{E})$ and

$$S^*_l \cap (E \setminus \overline{E}) = S_j \cap (E \setminus \overline{E}) \text{ for some } j \in J \setminus \overline{J}.$$

We calculate α_j vectors for all unselected sets based on formula (3.5) and select a best set using Definition 3.3.2 . Thus, by Definition 3.2.6, we obtain

$$\alpha_{j^*} = \begin{pmatrix} \alpha^1_{j^*} \\ \vdots \\ \alpha^p_{j^*} \end{pmatrix} \underset{mo}{\succeq} \begin{pmatrix} \lambda^*_1 \frac{c^1_l}{|S^*_l \cap (E \setminus \overline{E})|} \\ \vdots \\ \lambda^*_p \frac{c^p_l}{|S^*_l \cap (E \setminus \overline{E})|} \end{pmatrix} \quad \text{for all } l \in J \setminus \overline{J}. \tag{3.21}$$

Case 2: Let $l \notin J \setminus \overline{J}$, that is, all items in S^*_l are covered by Algorithm 2. In this case $S^*_l \cap (E \setminus \overline{E}) = \emptyset$ and $\frac{c^q_l}{|S^*_l \cap (E \setminus \overline{E})|} = \infty$. Thus we obtain

$$\begin{pmatrix} \alpha^1_{j^*} \\ \vdots \\ \alpha^p_{j^*} \end{pmatrix} \underset{mo}{\succeq} \begin{pmatrix} \lambda^*_1 \frac{c^1_l}{|S^*_l \cap (E \setminus \overline{E})|} \\ \vdots \\ \lambda^*_p \frac{c^p_l}{|S^*_l \cap (E \setminus \overline{E})|} \end{pmatrix} \quad \text{for all } l \notin J \setminus \overline{J}. \tag{3.22}$$

From (3.21) and (3.22) we conclude that $\alpha^*_j \underset{mo}{\succeq} \alpha_l$ for $l = 1, 2, \ldots, r^*$, or equivalently

$$\begin{pmatrix} \alpha^1_{j^*} \\ \vdots \\ \alpha^p_{j^*} \end{pmatrix} \underset{mo}{\succeq} \begin{pmatrix} \lambda^*_1 \frac{c^1_l}{|S^*_l \cap (E \setminus \overline{E})|} \\ \vdots \\ \lambda^*_p \frac{c^p_l}{|S^*_l \cap (E \setminus \overline{E})|} \end{pmatrix} \quad \text{for } l = 1, 2, \ldots, r^*,$$

and also

$$\begin{pmatrix} \alpha^1_{j^*} |S^*_l \cap (E \setminus \overline{E})| \\ \vdots \\ \alpha^p_{j^*} |S^*_l \cap (E \setminus \overline{E})| \end{pmatrix} \underset{mo}{\succeq} \begin{pmatrix} \lambda^*_1 c^1_l \\ \vdots \\ \lambda^*_p c^p_l \end{pmatrix} \quad \text{for } l = 1, 2, \ldots, r^*. \tag{3.23}$$

Now suppose that $\alpha^{k_1}_{j^*} |S^*_l \cap (E \setminus \overline{E})|$ is the maximum component of the left-hand-side vector in (3.23) and $\lambda^*_{k_2} c^{k_2}_l$ is the maximum component of the right-hand-side vector in (3.23) for some $k_1, k_2 \in \{1, 2, \ldots, p\}$, respectively. That is, $\alpha^{k_1}_{j^*} |S^*_l \cap (E \setminus \overline{E})| \leq \lambda^*_{k_2} c^{k_2}_l$ for some $k_1, k_2 \in \{1, 2, \ldots, p\}$ and for all $l = 1, 2, \ldots, p$. Then also $\alpha^k_{j^*} |S^*_l \cap (E \setminus \overline{E})| \leq \alpha^{k_1}_{j^*} |S^*_l \cap (E \setminus \overline{E})|$

and

$$\alpha_{j^*}^k |S_l^* \cap (E \setminus \overline{E})| \leq \lambda_{k_2}^* c_l^{k_2} \text{ for } k = 1, 2, \ldots, p \text{ and } l = 1, 2, \ldots, r^*. \tag{3.24}$$

By setting $k = 1,\ k = 2, \ldots, k = k_1,\ \ldots,\ k = k_2,\ \ldots\ k = p$ in inequality (3.24) and defining $c_{\max}^{k_2}$ and $c_{\min}^{k_1}$ as in (3.19), we obtain the following inequalities for $l = 1, 2 \ldots, r^*$.

$$\begin{array}{lllll}
\alpha_{j^*}^1 |S_l^* \cap (E \setminus \overline{E})| \leq \frac{\lambda_{k_2}^* c_l^{k_2}}{c_l^1} c_l^1 & \leq \lambda_{k_2}^* \lceil \frac{c_l^{k_2}}{c_l^1} \rceil c_l^1 & \leq \lambda_{k_2}^* \lceil \frac{c_{\max}^{k_2}}{c_{\min}^1} \rceil c_l^1 \\
\alpha_{j^*}^2 |S_l^* \cap (E \setminus \overline{E})| \leq \frac{\lambda_{k_2}^* c_l^{k_2}}{c_l^2} c_l^2 & \leq \lambda_{k_2}^* \lceil \frac{c_l^{k_2}}{c_l^2} \rceil c_l^2 & \leq \lambda_{k_2}^* \lceil \frac{c_{\max}^{k_2}}{c_{\min}^2} \rceil c_l^2 \\
\vdots \qquad\qquad \vdots & \vdots & \vdots \\
\alpha_{j^*}^{k_1} |S_l^* \cap (E \setminus \overline{E})| \leq \frac{\lambda_{k_2}^* c_l^{k_2}}{c_l^{k_1}} c_l^{k_1} & \leq \lambda_{k_2}^* \lceil \frac{c_l^{k_1}}{c_l^{k_1}} \rceil c_l^{k_1} & \leq \lambda_{k_2}^* \lceil \frac{c_{\max}^{k_2}}{c_{\min}^{k_1}} \rceil c_l^{k_1} \\
\vdots \qquad\qquad \vdots & \vdots & \vdots \\
\alpha_{j^*}^{k_2} |S_l^* \cap (E \setminus \overline{E})| \leq \lambda_{k_2}^* c_l^{k_2} & & \\
\vdots \qquad\qquad \vdots & \vdots & \vdots \\
\alpha_{j^*}^p |S_l^* \cap (E \setminus \overline{E})| \leq \frac{\lambda_{k_2}^* c_l^{k_2}}{c_l^p} c_l^p & \leq \lambda_{k_2}^* \lceil \frac{c_l^{k_2}}{c_l^p} \rceil c_l^p & \leq \lambda_{k_2}^* \lceil \frac{c_{\max}^{k_2}}{c_{\min}^p} \rceil c_l^p.
\end{array}$$

We define $\delta_{k_2} = \max\limits_{k=1,2,\ldots,p,\ k \neq k_2} \{\lambda_{k_2}^* \lceil \frac{c_{\max}^{k_2}}{c_{\min}^k} \rceil\}$. Then we get

$$\begin{pmatrix} \alpha_{j^*}^1 |S_l^* \cap (E \setminus \overline{E})| \\ \alpha_{j^*}^2 |S_l^* \cap (E \setminus \overline{E})| \\ \vdots \\ \alpha_{j^*}^{k_1} |S_l^* \cap (E \setminus \overline{E})| \\ \vdots \\ \alpha_{j^*}^{k_2} |S_l^* \cap (E \setminus \overline{E})| \\ \vdots \\ \alpha_{j^*}^p |S_l^* \cap (E \setminus \overline{E})| \end{pmatrix} \leqq \begin{pmatrix} \lambda_{k_2}^* \lceil \frac{c_{\max}^{k_2}}{c_{\min}^1} \rceil c_l^1 \\ \lambda_{k_2}^* \lceil \frac{c_{\max}^{k_2}}{c_{\min}^2} \rceil c_l^2 \\ \vdots \\ \lambda_{k_2}^* \lceil \frac{c_{\max}^{k_2}}{c_{\min}^{k_1}} \rceil c_l^{k_1} \\ \vdots \\ \lambda_{k_2}^* c_l^{k_2} \\ \vdots \\ \lambda_{k_2}^* \lceil \frac{c_{\max}^{k_2}}{c_{\min}^p} \rceil c_l^p \end{pmatrix} \leqq \delta_{k_2} \begin{pmatrix} c_l^1 \\ c_l^2 \\ \vdots \\ c_l^{k_1} \\ \vdots \\ c_l^{k_2} \\ \vdots \\ c_l^p \end{pmatrix} \quad \text{for} \ \ l = 1, 2, \ldots, r^*. \tag{3.25}$$

Since there are p choices for k_2 in inequality (3.25), we have δ_{k_2} for $k_2 = 1, 2, \ldots, p$. For

$k_2 = 1$, inequality (3.25) can be written as:

$$\begin{pmatrix} \alpha^1_{j^*}|S^*_l \cap (E \setminus \overline{E})| \\ \alpha^2_{j^*}|S^*_l \cap (E \setminus \overline{E})| \\ \vdots \\ \alpha^p_{j^*}|S^*_l \cap (E \setminus \overline{E})| \end{pmatrix} \leqq \delta_1 \begin{pmatrix} c^1_l \\ c^2_l \\ \vdots \\ c^p_l \end{pmatrix} \text{for } l = 1, 2, \ldots, r^*, \tag{3.26}$$

where $\delta_1 = \max\limits_{k=1,2,\ldots,p,\ k\neq 1} \{\lambda^*_1 \lceil \frac{c^1_{\max}}{c^k_{\min}} \rceil\}$. For $k_2 = 2$, inequality (3.25) can be written as:

$$\begin{pmatrix} \alpha^1_{j^*}|S^*_l \cap (E \setminus \overline{E})| \\ \alpha^2_{j^*}|S^*_l \cap (E \setminus \overline{E})| \\ \vdots \\ \alpha^p_{j^*}|S^*_l \cap (E \setminus \overline{E})| \end{pmatrix} \leqq \delta_2 \begin{pmatrix} c^1_l \\ c^2_l \\ \vdots \\ c^p_l \end{pmatrix} \text{ for } l = 1, 2, \ldots, r^*, \tag{3.27}$$

where $\delta_2 = \max\limits_{k=1,2,\ldots,p,\ k\neq 2} \{\lambda^*_2 \lceil \frac{c^2_{\max}}{c^k_{\min}} \rceil\}$. By continuing this process for $k_2 = p$, inequality (3.25) becomes

$$\begin{pmatrix} \alpha^1_{j^*}|S^*_l \cap (E \setminus \overline{E})| \\ \alpha^2_{j^*}|S^*_l \cap (E \setminus \overline{E})| \\ \vdots \\ \alpha^p_{j^*}|S^*_l \cap (E \setminus \overline{E})| \end{pmatrix} \leqq \delta_p \begin{pmatrix} c^1_l \\ c^2_l \\ \vdots \\ c^p_l \end{pmatrix} \text{ for } l = 1, 2, \ldots, r^*, \tag{3.28}$$

where $\delta_p = \max\limits_{k=1,2,\ldots,p,\ k\neq p} \{\lambda^*_p \lceil \frac{c^p_{\max}}{c^k_{\min}} \rceil\}$.
Let $\delta = \max\limits_{k_2=1,2,\ldots,p} \left\{ \max\limits_{\substack{k=1,2,\ldots,p,\\ k\neq k_2}} \{\lambda^*_{k_2} \lceil \frac{c^{k_2}_{\max}}{c^k_{\min}} \rceil\} \right\} = \max\limits_{\substack{k_1,k_2=1,2,\ldots,p,\\ k_1\neq k_2}} \{\lambda^*_{k_2} \lceil \frac{c^{k_2}_{\max}}{c^{k_1}_{\min}} \rceil\}$. Then inequality (3.28) becomes

$$\begin{pmatrix} \alpha^1_{j^*}|S^*_l \cap (E \setminus \overline{E})| \\ \alpha^2_{j^*}|S^*_l \cap (E \setminus \overline{E})| \\ \vdots \\ \alpha^p_{j^*}|S^*_l \cap (E \setminus \overline{E})| \end{pmatrix} \leqq \delta \begin{pmatrix} c^1_l \\ c^2_l \\ \vdots \\ c^p_l \end{pmatrix} \quad \text{for} \quad l = 1, 2, \ldots, r^*. \tag{3.29}$$

Summing over all sets in the weak Pareto cover, inequality (3.29) gives the following:

$$\begin{pmatrix} \sum_{l=1}^{r^*} \alpha^1_{j^*}|S^*_l \cap (E \setminus \overline{E})| \\ \sum_{l=1}^{r^*} \alpha^2_{j^*}|S^*_l \cap (E \setminus \overline{E})| \\ \vdots \\ \sum_{l=1}^{r^*} \alpha^p_{j^*}|S^*_l \cap (E \setminus \overline{E})| \end{pmatrix} \leqq \delta \begin{pmatrix} \sum_{l=1}^{r^*} c^1_l \\ \sum_{l=1}^{r^*} c^2_l \\ \vdots \\ \sum_{l=1}^{r^*} c^p_l \end{pmatrix} = \delta \begin{pmatrix} z_1(x^*) \\ z_2(x^*) \\ \vdots \\ z_p(x^*) \end{pmatrix}. \tag{3.30}$$

Using inequality (3.20), inequality (3.30) can be written as follows:

$$\begin{pmatrix} \alpha^1_{j^*}|(E \setminus \overline{E})| \\ \alpha^2_{j^*}|(E \setminus \overline{E})| \\ \vdots \\ \alpha^p_{j^*}|(E \setminus \overline{E})| \end{pmatrix} \leqq \delta \begin{pmatrix} z_1(x^*) \\ z_2(x^*) \\ \vdots \\ z_p(x^*) \end{pmatrix}, \tag{3.31}$$

where $\delta = \max\limits_{\substack{k_1,k_2=1,2,\ldots,p, \\ k_1 \neq k_2}} \{\lambda^*_{k_2} \lceil \frac{c^{k_2}_{\max}}{c^{k_1}_{\min}} \rceil\}$. In the iteration in which the item e_k is covered, the number of uncovered items is $(m - k + 1)$. That is, $|(E \setminus \overline{E})| = m - k + 1$. Thus, inequality (3.31) becomes:

$$\begin{pmatrix} \alpha^1_{j^*} \\ \alpha^2_{j^*} \\ \vdots \\ \alpha^p_{j^*} \end{pmatrix} \leqq \delta \begin{pmatrix} z_1(x^*)/(m - k + 1) \\ z_2(x^*)/(m - k + 1) \\ \vdots \\ z_p(x^*)/(m - k + 1) \end{pmatrix}. \tag{3.32}$$

Since $p(e_k) = \alpha_{j^*}$, from inequality (3.32) we conclude

$$p(e_k) \leqq \delta \begin{pmatrix} z_1(x^*)/(m-k+1) \\ z_2(x^*)/(m-k+1) \\ \vdots \\ z_p(x^*)/(m-k+1) \end{pmatrix}.$$

□

Corollary 16 shows that the cost of the cover associated with the solution yield by Algorithm 2 is equal to the cost of covering all items.

Corollary 16. *If $\bar{x} \in X$ is a solution of the MOSCP yield by Algorithm 2 for $\lambda^* \in \mathcal{R}^p_>$, then*

$$[\lambda_1^* z_1(\bar{x}), \lambda_2^* z_2(\bar{x}), \ldots, \lambda_p^* z_p(\bar{x})]^T = \sum_{k=1}^{m} p(e_k).$$

Proof. In Algorithm 2, the cost of each set selected in each iteration is distributed among the items covered in that iteration. Therefore, the cost of covering all items, at the termination of Algorithm 2, is equal to $\sum_{k=1}^{m} p(e_k)$. On the other hand, the cost of the selected cover is given by the objective vector of problem (3.3), $[\lambda_1^* z_1(\bar{x}), \lambda_2^* z_2(\bar{x}), \ldots, \lambda_p^* z_p(\bar{x})]^T$. By definition, a cover covers all items and thus, we have, $[\lambda_1^* z_1(\bar{x}), \lambda_2^* z_2(\bar{x}), \ldots, \lambda_p^* z_p(\bar{x})]^T = \sum_{k=1}^{m} p(e_k)$. □

We analyze the approximation factor $(1+\epsilon)$ for the MOSCP using the result in Corollary 16. We show that $\bar{x}$ which is the solution yield by Algorithm 2 is a solution such that $z(\bar{x}) \leqq (1+\epsilon) z(x^*)$ where x^* is the optimal solution of problem (3.3) associated with $\lambda^* \in \mathcal{R}^p_>$. Further, we show that the approximation factor, $(1+\epsilon)$, depends on the maximum and minimum cost coefficients of each objective function, the weight vector and the number of items of the given test instance. Corollary 16 yields Theorem 3.3.2 which is the main result of this research.

Theorem 3.3.2. *Let $\bar{x} \in X$ be a solution of the MOSCP yield by Algorithm 2 and x^* be an optimal solution of problem (3.3) associated with $\lambda^* \in \mathcal{R}^p_>$. Then*

$$\begin{pmatrix} z_1(\bar{x}) \\ z_2(\bar{x}) \\ \vdots \\ z_p(\bar{x}) \end{pmatrix} \leqq \delta \log(m)/\lambda^*_{\min} \begin{pmatrix} z_1(x^*) \\ z_2(x^*) \\ \vdots \\ z_p(x^*) \end{pmatrix},$$

where $\delta = \max\limits_{\substack{k_1,k_2=1,2,\ldots,p,\\ k_1\neq k_2}} \{\lambda^*_{k_2} \lceil \frac{c^{k_2}_{\max}}{c^{k_1}_{\min}} \rceil\}$ *and* $\lambda^*_{\min} = \min\{\lambda^*_1, \lambda^*_2, \ldots, \lambda^*_p\}$.

Proof. Using Lemma 3.3.2, and summing over all items we get,

$$\sum_{k=1}^{m} p(e_k) \leq \sum_{k=1}^{m} \delta \begin{pmatrix} z_1(x^*)/(m-k+1) \\ z_2(x^*)/(m-k+1) \\ \vdots \\ z_p(x^*)/(m-k+1) \end{pmatrix} = (1 + 1/2 + \cdots + 1/m)\delta \begin{pmatrix} z_1(x^*) \\ z_2(x^*) \\ \vdots \\ z_p(x^*) \end{pmatrix},$$

where $\delta = \max\limits_{\substack{k_1,k_2=1,2,\ldots,p,\\ k_1\neq k_2}} \{\lambda^*_{k_2} \lceil \frac{c^{k_2}_{\max}}{c^{k_1}_{\min}} \rceil\}$. As $(1 + 1/2 + \cdots + 1/m) \approx \log m$, we obtain

$$\sum_{k=1}^{m} p(e_k) \leqq \delta \log(m)[z_1(x^*), z_2(x^*), \ldots, z_p(x^*)]^T \tag{3.33}$$

and using Corollary 16

$$\begin{pmatrix} \lambda^*_1 z_1(\bar{x}) \\ \lambda^*_2 z_2(\bar{x}) \\ \vdots \\ \lambda^*_p z_p(\bar{x}) \end{pmatrix} \leqq \delta \log(m) \begin{pmatrix} z_1(x^*) \\ z_2(x^*) \\ \vdots \\ z_p(x^*) \end{pmatrix}. \tag{3.34}$$

Let $\lambda^*_{min} = \min\{\lambda^*_1, \lambda^*_2, \ldots, \lambda^*_p\}$, then inequality (3.34) gives the following:

$$\lambda^*_{min} \begin{pmatrix} z_1(\bar{x}) \\ z_2(\bar{x}) \\ \vdots \\ z_p(\bar{x}) \end{pmatrix} \leqq \begin{pmatrix} \lambda^*_1 z_1(\bar{x}) \\ \lambda^*_2 z_2(\bar{x}) \\ \vdots \\ \lambda^*_p z_P(\bar{x}) \end{pmatrix} \leqq \delta \log(m) \begin{pmatrix} z_1(x^*) \\ z_2(x^*) \\ \vdots \\ z_p(x^*) \end{pmatrix}.$$

Thus we have,

$$\begin{pmatrix} z_1(\bar{x}) \\ z_2(\bar{x}) \\ \vdots \\ z_p(\bar{x}) \end{pmatrix} \leqq \delta \log(m)/\lambda^*_{min} \begin{pmatrix} z_1(x^*) \\ z_2(x^*) \\ \vdots \\ z_p(x^*) \end{pmatrix},$$

where $\delta = \max\limits_{\substack{k_1,k_2=1,2,\ldots,p, \\ k_1 \neq k_2}} \{\lambda^*_{k_2} \lceil \frac{c^{k_2}_{\max}}{c^{k_1}_{\min}} \rceil\}$. This completes the proof. □

Now we prove that Algorithm 2 can be used to obtain an $(1+\epsilon)$-approximate Pareto set P^ϵ for the MOSCP. That is, for any weak Pareto outcome, Algorithm 2 yields an $(1+\epsilon)$-approximate Pareto outcome. Let $(1+\epsilon) = \delta \log(m)/\lambda^*_{\min} > 0$. Corollary 17 shows that when running for all possible λ vectors, Algorithm 2 can be used to approximate all weak Pareto outcomes of the MOSCP with the bound $(1+\epsilon)$.

Corollary 17. *For every $x \in X_{wE}$ of the MOSCP, there exist a weight vector $\lambda \in \mathcal{R}^p_>$ and a solution $\bar{x} \in X$ yield by Algorithm 2 satisfying the following condition:*

$$\begin{pmatrix} z_1(\bar{x}) \\ z_2(\bar{x}) \\ \vdots \\ z_p(\bar{x}) \end{pmatrix} \leqq (1+\epsilon) \begin{pmatrix} z_1(x) \\ z_2(x) \\ \vdots \\ z_p(x) \end{pmatrix}, \qquad (3.35)$$

where

$$\epsilon = \delta \log(m)/\lambda_{\min},\ \delta = \max_{\substack{k_1,k_2=1,2,\dots,p,\\ k_1\neq k_2}} \{\lambda_{k_2} \lceil \frac{c^{k_2}_{\max}}{c^{k_1}_{\min}} \rceil\} \text{ and } \lambda_{\min} = \min\{\lambda_1,\lambda_2,\dots,\lambda_p\}. \quad (3.36)$$

Proof. Let $x \in X_{wE}$. Then, by Proposition 3.2.2, there exists $\lambda \in \mathcal{R}^p_>$ such that x is an optimal solution of problem (3.3). We show that there exists a solution $\bar{x} \in X$ such that (3.35) holds. Suppose that Algorithm 2 is executed for λ and returns a cover associated with the solution $\bar{x}$. Then by Theorem 3.3.2, we have $\begin{pmatrix} z_1(\bar{x}) \\ z_2(\bar{x}) \\ \vdots \\ z_p(\bar{x}) \end{pmatrix} \leqq \delta \log(m)/\lambda_{\min} \begin{pmatrix} z_1(x) \\ z_2(x) \\ \vdots \\ z_2(x) \end{pmatrix}$ and (3.35) holds. $\square$

According to Corollary 17, the bound on every component of a weak Pareto solution associated with λ is $\delta \log(m)/\lambda_{\min}$. Using condition (3.35) and Definition 3.2.7, we obtain the following theoretical result.

If Algorithm 2 is run for all $\lambda \in \mathcal{R}^P_{\geq}$, for every solution in the set $P_w(Y)$ there is a solution in the set of solutions returned by Algorithm 2 satisfying condition (3.35). Thus, in view of Definition 3.2.7, Algorithm 2 returns a $(1+\epsilon)$-approximate weak Pareto set of the MOSCP, where

$$1+\epsilon = \max_{\lambda \in \mathcal{R}^p_{\geq}} \{\delta \log(m)/\lambda_{\min}\},\ \delta \text{ given in (3.36)},\ m \text{ is the number of items, and } \lambda \in \mathcal{R}^p_{\geq}. \quad (3.37)$$

3.3.2.3 Complexity

In this section we show that the running times of Algorithm 1 and Algorithm 2 are polynomial, if they are run for a fixed number of $\lambda \in \mathcal{R}^p_{\geq}$.

Theorem 3.3.3. *Let $\lambda \in \mathcal{R}^p_{\geq}$ be fixed. Algorithm 1 and Algorithm 2 are polynomial-time* *algorithms.*

Proof. In the generic algorithm the loop in the main step iterates for $O(m)$ time, where $|E| = m$.
In Procedure 1, the minimum of α_j for $j = 1, 2, \ldots, n$ can be found in $O(\log n)$ time using a priority heap ([3]). Therefore the total running time of Algorithm 1 is $O(m \log n)$ and thus Algorithm 1 is a polynomial-time algorithm.

In Procedure 2, the maximum component of each cost effectiveness vector α_j for $j = 1, 2, \ldots, n$ can be found in $O(\log p)$ time using a priority heap where p is the number of objectives ([3]). Then we get the minimum of these maximum components in constant time since we update the minimum value as we calculate each maximum. Therefore the total running time of Algorithm 2 is $O(n \log p)$ and thus Algorithm 2 is a polynomial-time algorithm. □

3.4 Computational results

This section presents the computational results obtained with Algorithm 1 and Algorithm 2 on a variety of BOSCP test instances that are generated by Gandibleux ([1]). These algorithms were implemented using MATLAB interface while all experiments were carried out on a Dell Vostro 1400 computer with a Pentium- IV and 2 GB RAM. The test instances are divided into four different groups A, B, C, and D according to the type of objective functions. For the instances of type A, the cost coefficients of each objective function are generated randomly and independently. For the instances of type B, the cost coefficients of the first objective function are generated randomly and independently, and the coefficients of the second objective function have a relation with those of the first objective function. For the type C test instances, the sets of each test instance are divided into subgroups and each set in one subgroup has the same cost coefficients for both objective functions. The type D test instances have characteristics of type B and type C. We report the computational results for the test instances listed in Table 3.1, which we consider the most representative for our work.

Name of the test instance	m, number of items	n, number of sets
2scp41A	40	200
2scp61D	60	600
2scp81C	80	800
2scp201B	200	1000

Table 3.1: Characteristics of the BOSCP test instances

3.4.1 Measures for evaluating the performance of algorithms

Comparing the $(1+\epsilon)$-approximate Pareto set with the Pareto set is important to assess the quality of the approximation. Several criteria are available in the literature to evaluate the quality of approximate Pareto sets ([11], [53]).

In this study, we measure the quality of the $(1+\epsilon)$-approximate Pareto set in two different ways. For the first measure, we calculate the relative deviation of the range of each objective function values of the BOSCP returned by Algorithm 1 and Algorithm 2 with respect to the exact range. The range for an objective function is obtained by finding the minimum and the maximum values of the function over the computed set while the exact range is obtained by using the Pareto set.

The second type of measures we propose is calculated using the computed approximate Pareto sets. These measures are constructed to resemble the meaning of the theoretical factors of the algorithms given in Corollaries 15 and 17.

Let Algorithm 1 (or Algorithm 2) be run for $\lambda \in \mathcal{R}^p_>$ and return a solution $\bar{x}$. Let x^* be a supported (or weakly efficient) solution associated with the weight λ. Then $z(\bar{x}) \in P^\epsilon_s$ (or P^ϵ_w) and $z(x^*) \in P_s(Y)$ (or $P_w(Y)$). Based on inequality (3.17), the theoretical factor

$1+\epsilon = \log m$ of Algorithm 1 satisfies

$$\frac{\sum_{q=1}^{p} \lambda_q z_q(\bar{x})}{\sum_{q=1}^{p} \lambda_q z_q(x^*)} \le \log m.$$

We propose to calculate experimental factor of Algorithm 1 as follows:

$$\max_{\lambda \in \Lambda} \left\{ \frac{\sum_{q=1}^{p} \lambda_q z_q(\bar{x})}{\sum_{q=1}^{p} \lambda_q z_q(x^*)} \right\} \tag{3.38}$$

for $z(\bar{x}) \in P_s^{\epsilon},\ z(x^*) \in P_s(Y)$, and $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{100}\}$ is the set of weights used for the computation. Based on inequality (3.35), the theoretical factor $1+\epsilon$ of Algorithm 2 satisfies

$$\frac{z_q(\bar{x})}{z_q(x^*)} \le \delta \log m / \lambda_{min}$$

for $q = 1, 2, \ldots, p$. We propose to calculate the experimental factor of Algorithm 2 as follows:

$$\max_{q=1,2,\ldots,p} \left\{ \frac{\max\limits_{z(\bar{x}) \in P_w^{\epsilon}} \{z_q(\bar{x})\}}{\min\limits_{z(x^*) \in P_w(Y)} \{z_q(x^*)\}} \right\}. \tag{3.39}$$

For simplicity, we normalize the λ vectors and have $\sum_{q=1}^{p} \lambda_q = 1$ in all our experiments for $\lambda \in \mathcal{R}^p_>$.

In our analysis we do not include computational times because they strongly depend on the type of computer. For example, Algorithm 2 takes 2 minutes to construct a $(1+\epsilon)$-approximate weak Pareto set of the instance 2scp201B (one of the largest test instances) on the above mentioned Dell Vostro 1400 computer. When a faster computer is used, it takes only 2 seconds.

3.4.2 Results

Table 3.2 shows the ranges of the objective functions, z_1 and z_2, for the test instances listed in Table 3.1. We obtain these ranges for the exact Pareto set, $(1+\epsilon)$-approximate supported Pareto set when running Algorithm 1, and $(1+\epsilon)$-approximate weak Pareto set when running Algorithm 2. The exact Pareto outcomes are computed by applying the augmented ϵ-constraint method ([29]) and the solver Cplex 12.4 with MATLAB interface.

In Table 3.2, the first column gives the name of the test instance while the second

Instance	Name	Exact	Algorithm 1	Algorithm 2
2scp41A	z_1	(880, 2647)	(895, 3273)	(3976, 7021)
	z_1	(0, 1)	(0.017, 1.23)	(3.52, 2.65)
	z_2	(888, 2498)	(995, 3376)	(4566, 7103)
	z_2	(0, 1)	(0.120, 1.35)	(4.14, 2.84)
2scp61D	z_1	(1790, 9064)	(1921, 8704)	(19804, 28178)
	z_1	(0, 1)	(0.732, -0.397)	(10.063, 2.109)
	z_2	(1695, 8759)	(1828, 9699)	(13461, 28660)
	z_2	(0, 1)	(0.078, 0.107)	(6.942, 2.272)
2scp81C	z_1	(908, 5089)	(1036, 1151)	(32116, 53708)
	z_1	(0, 1)	(0.141, -0.774)	(34.370, 9.554)
	z_2	(1372, 4578)	(2105, 9794)	(20151, 58757)
	z_2	(0, 1)	(0.534, 1.139)	(13.687, 11.835)
2scp201B	z_1	(1314, 18853)	(1514, 19481)	(76751, 121100)
	z_1	(0, 1)	(0.152, 0.033)	(57.410, 5.423)
	z_2	(1189, 18397)	(1386, 19870)	(7567, 125400)
	z_2	(0, 1)	(0.166, 0.080)	(5.364, 5.816)

Table 3.2: Ranges of z_1 and z_2 computed by each algorithm

column specifies the objective function. Columns three, four, and five give ranges for an objective function obtained by finding the minimum and the maximum values of the function over the computed set. For example, the first objective function, z_1, has the minimum value of 880 and the maximum value of 2647 on the exact Pareto set of the 2scp41A test instance. Thus the exact range of z_1 of this test instance is $(880, 2647)$. For simplicity, in the next row, we normalize the range $(880, 2647)$ to the interval $(0, 1)$. The fourth and the fifth columns also indicate the relative deviation of each objective function values returned by Algorithm 1 and Algorithm 2 with respect to the exact range. For example, consider the 2scp41A test instance. The range of z_1 according to Algorithm 1 is $(895, 3273)$. That is, relative deviation of 895 with respect to 880 is 0.017 and that of 3273 with respect to 2647 is 1.23. Note that two instances, 2scp61D and 2scp81D, have negative relative deviations based on Algorithm 1. The negative numbers result from the component-wise comparison of the approximate Pareto outcome and corresponding exact Pareto outcome while Algorithm 1 yields approximate outcomes based on the weighted-sum preference relation. Based on the data in Table 3.2, we see that the ranges of z_1 and z_2 given by Algorithm 1 are very close to the exact ranges. On the other hand, the ranges given by Algorithm 2 are far away from the exact ranges.

Table 3.3 shows the theoretical and experimental approximation factors of the algorithms for the test instances listed in Table 3.1. The first column gives the name of the test instance. The second and third columns give the theoretical and experimental factors of Algorithm 1 while the fourth and fifth columns give the theoretical and experimental factors of Algorithm 2. These factors are obtained using inequalities (3.17) and (3.38), and (3.35) and (3.39) respectively. For example, the 2scp41A test instance has $m = 40$ items, $c^1_{\max} = 200$, $c^1_{\min} = 2$, $c^2_{\max} = 200$, and $c^1_{\min} = 3$. Further, we have $\lambda_1 + \lambda_2 = 1$ and $\lambda_{min} = 0.01$. The theoretical approximation factor of Algorithm 1 is $\log 40 = 1.602$

	Algorithm 1		Algorithm 2	
	Theoretical (3.17)	Experimental (3.38)	Theoretical (3.35)	Experimental (3.39)
2scp41A	1.602	1.116	15860.393	7.998
2scp61D	1.778	1.221	3243.143	16.906
2scp81C	1.903	1.167	24178.758	59.012
2scp201B	2.301	1.192	76085.858	104.491

Table 3.3: Theoretical and experimental approximation factors

and the experimental approximation factor is equal to 1.116 (by formulas (3.18) and (3.38), respectively). The theoretical approximation factor of Algorithm 2 is equal to 15860.393 (by formula (3.37)). The experimental approximation factor is calculated using (3.39).

The values $\max\{z_1(\bar{x})\}$, $\max\{z_2(\bar{x})\}$, $\min\{z_1(x^*)\}$ and $\min\{z_2(x^*)\}$ for $z(\bar{x}) \in P_w^\epsilon$ and $z(x^*) \in P_w(Y)$ are obtained from Table 3.2 and the experimental factor is equal to 7.998. As seen in Table 3.3, the computational results for all test instances obey the theoretical approximation factors. Additionally, although these theoretical factors are larger, the experimental factors are much smaller.

Figures 3.1, 3.3, 3.5 and 3.7 depict $(1+\epsilon)$-approximate supported Pareto sets obtained by Algorithm 1 while Figures 3.2, 3.4, 3.6 and 3.8 depict $(1+\epsilon)$-approximate weak Pareto sets obtained by Algorithm 2. Furthermore, in each figure either the (exact) supported Pareto set or the (exact) Pareto set is depicted for comparison. For example, Figure 3.1 shows the supported Pareto outcomes and the $(1+\epsilon)$-approximate supported Pareto outcomes obtained by Algorithm 1 for 2scp41A instance. Figure 3.2 shows the Pareto outcomes and the $(1+\epsilon)$-approximate weak Pareto outcomes obtained by Algorithm 2 for 2scp41A instance.

Based on Figures 3.1, 3.3, 3.5 and 3.7, we observe that the $(1+\epsilon)$-approximate supported Pareto outcomes returned by Algorithm 1 are very close to the supported Pareto outcomes and all supported Pareto outcomes are approximated. Further, as seen in Table 3.2, we observe why of 2scp61D and 2scp81C have negative relative deviations based on Figures 3.3 and 3.5. In agreement with Table 3.2, Figures 3.2, 3.4, 3.6 and 3.8 show that the $(1+\epsilon)$-approximate weak Pareto outcomes obtained by Algorithm 2 are worse than the Pareto outcomes.

Based on these computational results, we observe that the approximate outcomes obey the theoretical results of Section 3, and the experimental results are far better than the theoretical results.

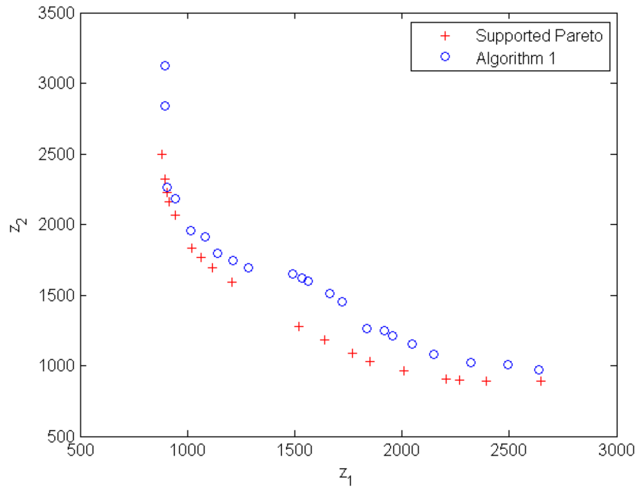


Figure 3.1: Comparison of outcomes obtained by Algorithm 1 to the supported Pareto outcomes for 2scp41A

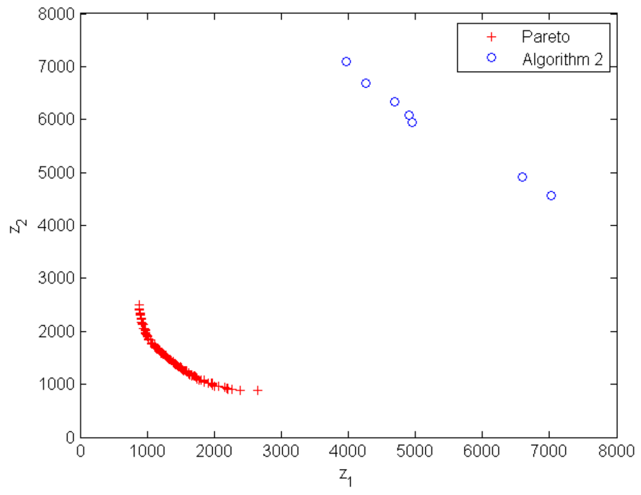


Figure 3.2: Comparison of outcomes obtained by Algorithm 2 to the Pareto outcomes for 2scp41A

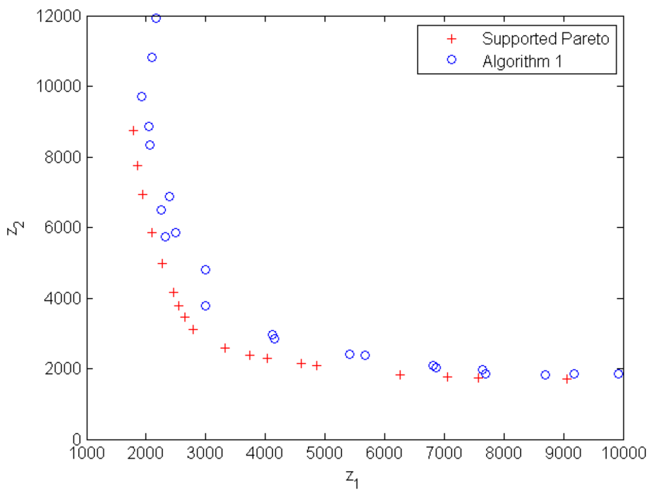


Figure 3.3: Comparison of outcomes obtained by Algorithm 1 to the supported Pareto outcomes for 2scp61D

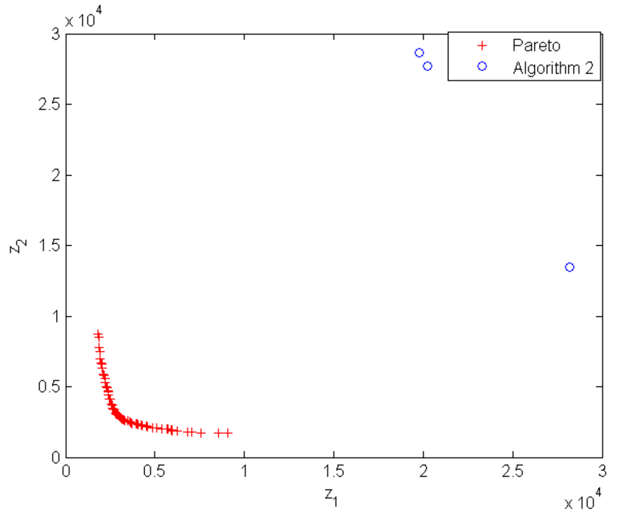


Figure 3.4: Comparison of outcomes obtained by Algorithm 2 to the Pareto outcomes for 2scp61D

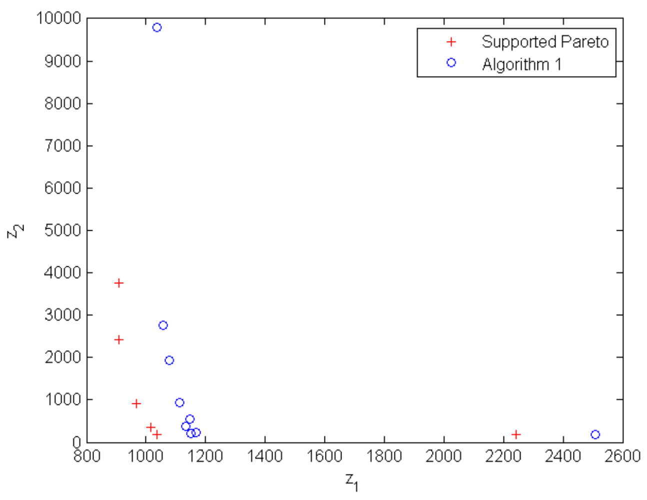


Figure 3.5: Comparison of outcomes obtained by Algorithm 1 to the supported Pareto outcomes for 2scp81C

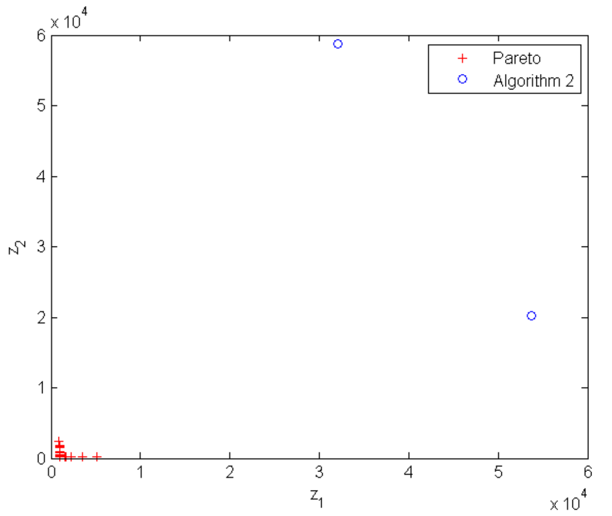


Figure 3.6: Comparison of outcomes obtained by Algorithm 2 to the Pareto outcomes for 2scp81C

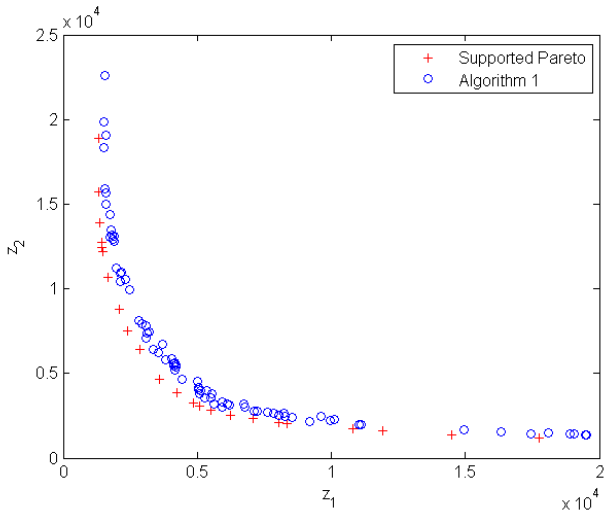


Figure 3.7: Comparison of outcomes obtained by Algorithm 1 to the supported Pareto outcomes for 2scp210B

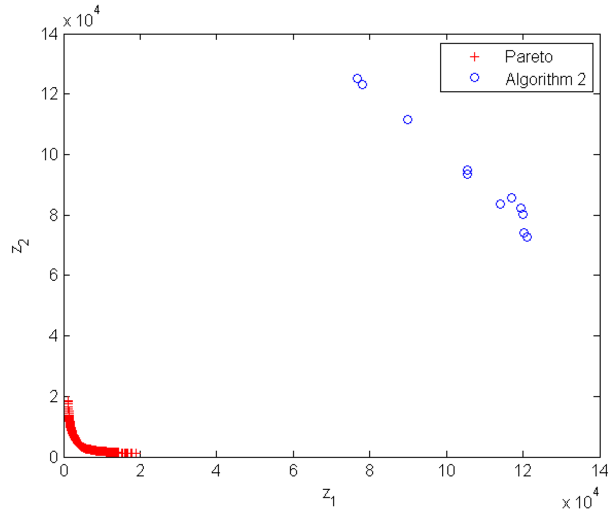


Figure 3.8: Comparison of outcomes obtained by Algorithm 2 to the Pareto outcomes for 2scp201B

3.5 Conclusion

We have developed two polynomial-time algorithms for approximating the Pareto set of the MOSCP and derived their approximation factors. Algorithm 1 approximates the supported Pareto set with a constant factor depending only on the number of items of the test instance. Algorithm 2 approximates the weak Pareto set with a factor depending on the problem data (the number of items and the magnitude of the cost coefficients) and also on the weight vector used for computing the approximation. While Algorithm 1 approximates only a subset of the Pareto set, its approximation factor is significantly smaller than that of Algorithm 2. We have applied the algorithms to BOSCPs of various sizes. For both algorithms, the obtained numerical results not only confirm but are far better than the theoretical results. Based on the theoretical and experimental results, Algorithm 1 is superior to Algorithm 2.

This study motivates the development of an algorithm approximating all Pareto points of the MOSCP with a better approximation factor than that of Algorithm 2. Alternatively, a phase-2 algorithm complementing the approximation of supported Pareto points by Algorithm 1 could be proposed. Additionally, the presented results show that λ vectors exist to approximate supported or weak Pareto sets of the MOSCP, but they do not reveal how these vectors can be found. Thus, an important avenue of future study is to investigate how these vectors can be constructed.

Chapter 4

Add-Improve Algorithm for approximating the Pareto set of the Multiobjective Set Covering Problem

4.1 Introduction

Multiobjective combinatorial optimization (MOCO) problems involve optimizing more than one objective function on a finite set of feasible solutions. Because there may not exist a single optimal solution to a MOCO problem as the objective functions are in conflict with each other, a solution set exists and is referred to as the efficient set. The image of the efficient set is defined as the Pareto set. During the last 20 years, many heuristic methods for solving MOCO problems have been proposed. Surveys summarizing those efforts are given by Ehrgott [24], Ehrgott and Gandibleux [26], Ulungu and Teghem [100], and others.

One of the well-known combinatorial optimization problems is the set covering problem (SCP). It originates from facility location problems and is in the category of NP-hard problems ([89]). An instance of the SCP consists of a finite set of items and a family of subsets of the items so that every item belongs to at least one of the subsets in the family. In the single objective version, each set in the family has a positive scalar cost. The goal of the single objective SCP (SOSCP) is to determine a subset of sets, named a *cover*, among the sets in the family, so that all items are included in at least one set in the subset and the total cost associated with the selected sets is minimized. If there are p scalar costs for each set in the family, then the problem turns into the multiobjective set covering problem (MOSCP). The goal of MOSCP is to determine a set of covers referred to as efficient such that the total cost associated with each cover are minimized.

The SOSCP is a well-studied problem and different methods, especially heuristics methods, have been proposed in the literature to address it ([13], [70], and others). Although the MOSCP has many real-life applications in the fields such as scheduling, facility location, designing reserve systems ([20], [33]), it has not received as much attention as the SOSCP and only a few studies are found in the literature. In 1993, [61] proposed a heuristic algorithm generating only one solution of the MOSCP. [96] develop a heuristic enumeration technique for solving the MOSCP with quadratic objective functions. Under the assumption of differentiability, the authors linearize the quadratic objective functions and use the Gomory cut technique to get the set of efficient solutions. Jaszkiewicz ([53], [52]) provides a comparative study of multiobjective metaheuristics for the biobjective SCP (BOSCP). In particular, nine well-known multiobjective metaheuristics are compared with a new algorithm called the Pareto memetic algorithm (PMA). The performance of the multiobjective metaheuristics for the BOSCP depends on the problem structure. [85] propose a heuristic based two-phase method (TPM) to find the Pareto set of the BOSCP. In the first phase, the scalarized SCP is solved with a heuristic to generate a subset of the Pareto set called the supported Pareto set. In the second phase, a heuristic algorithm searches for the Pareto points located between two supported Pareto points. This heuristic optimizes one objective function at a time and requires that this SCP be reformulated by Lagrangian relaxation. [64] adapt a very large-scale neighborhood search ([3]) for the MOSCP and compare average running times of the adaptation with the PMA and the TPM for the BOSCP. The performance of their algorithm also depends on the problem structure.

All the methods for solving the MOSCP are based on heuristic or metaheuristic approaches. Due to their nature, it can not be ascertained that one is superior to another. Further, when obtaining or improving solutions of the MOSCP, these approaches do not evaluate the quality of the solutions in the objective space since they do not estimate the cost of the associated covers. The goal of this paper is to investigate how this cost can be integrated in an algorithm and propose a new heuristic method for solving the MOSCP.

This paper proposes a two-phase heuristic method to approximate the Pareto set of the MOSCP. The method is developed using two scalarization methods, the weighted-sum method and the weighted-Chebyshev method ([29]). The solutions to the MOSCP are constructed iteratively selecting sets which aid to minimize the objective functions. The distinguishing feature of this method in comparison to the other methods in the literature is the process of selecting sets to obtain feasible solutions. As a preprocessing of the algorithm, the best set to cover each item with respect to the each objective function is determined. When constructing a feasible solution, if a given item is not covered before selecting the best set to cover it, a merit function is constructed to estimate the value of the objective functions if the best set has been used.

The paper is organized as follows. Section 4.2 provides the formulation of the MOSCP, introduces the terminology, and discusses the methods used. Section 4.3 explains preliminary concept of the proposed algorithm while Section 4.4 introduces the two phases of the algorithm. A comparison of the performance of the algorithm with the performance of the PMA proposed by Jaszkiewicz [53] and computational results on test problems are presented in Section 4.5. The paper is concluded in Section 4.6.

4.2 Problem formulation

Let $\mathcal{R}^p$ be a finite dimensional Euclidean vector space. For $y^1, y^2 \in \mathcal{R}^p$, $y^1 \leq y^2$ denotes that $y_k^1 \leq y_k^2$ for all $k = 1, 2, \ldots, p$ and $y^1 \neq y^2$. The nonnegative orthant of $\mathcal{R}^p$ is defined as $\mathcal{R}^p_{\geq} = \{y \in \mathcal{R}^p : \ y \geq \underline{0}\}$.

In the SCP there is a set of m items, $E = \{e_1, e_2, \ldots, e_m\}$, with the index set $I = \{i : i = 1, 2, \ldots, m\}$, and a set of n subsets of E, $S = \{S_1, S_2, \ldots, S_n\}$, with the index set of $J = \{j \ : \ j = 1, 2, \ldots, n\}$. The items are grouped into subsets of E and an item e_i in E is said to be covered by a set S_j in S provided e_i in S_j. An instance of the SCP is given by the sets E and S. The binary coefficient a_{ij}, for $i \in I$ and $j \in J$, is equal to 1 if the item e_i is covered by the set S_j, and is equal to zero otherwise. A cover is defined as a subcollection $\{S_j : j \in J^* \subseteq J\}$ which is a subset of S such that all items of E are covered, where J^* is the index set of selected sets for the subcollection.

As mentioned in the Introduction, a feasible solution of the SCP requires that each item be covered by at least one selected set. Let $x \in \mathbb{Z}^n$ be the decision variable defined as follows,

$$x_j = \begin{cases} 1 & \text{if } S_j \text{ is selected for a cover} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } j \in J.$$

The set X of all feasible solutions is defined as

$$X = \{x \in \mathbb{Z}^n : \sum_{j \in J} a_{ij} x_j \geq 1 \text{ for } i \in I \text{ and } x_j \in \{0, 1\} \text{ for } j \in J\}.$$

Note that every feasible vector $x \in X$ is associated with a cover and vice versa.

The MOSCP has p conflicting objective functions, $z_q : \mathbb{Z}^n \to \mathbb{Z}$, with the index set $Q = \{q : q = 1, 2, \ldots, p\}$. Let $c_j^q > 0$ denote the cost of the set S_j with respect to the objective q for $q \in Q$. The cost of a feasible cover, $x \in X$, with respect to the objective q is $z_q(x) = \sum_{j \in J^*} c_j^q$. The set of all attainable outcomes, Y, for all feasible solutions, $x \in X$, is obtained by evaluating the p objective functions. That is $Y := z(X) \subset \mathcal{R}^p$, where $z = (z_1, \ldots, z_p)$.

The goal of the MOSCP is to find a cover such that the costs with respect to all objective functions are minimized. The MOSCP can be presented as follows:

$$\min\, z(x) = \left[z_1(x) = \sum_{j=1}^{n} c_j^1 x_j,\quad z_2(x) = \sum_{j=1}^{n} c_j^2 x_j,\quad \ldots,\quad z_p(x) = \sum_{j=1}^{n} c_j^p x_j \right] \qquad \text{subject to } x \in X. \tag{4.1}$$

It is of interest to define efficient solutions for the MOSCP.

Definition 4.2.1. *A solution $x^* \in X$ is called an efficient solution of the MOSCP if there does not exist a solution $x \in X$ such that $z(x) \leq z(x^*)$.*

The set of all efficient solutions is denoted by X_E. The image $z(x) \in Y$ of an efficient solution $x \in X_E$ is called a Pareto outcome. The image, $z(X_E)$, of X_E is denoted by Y_P and is referred to as the Pareto set. Solving the MOSCP is understood as finding its efficient solutions and Pareto outcomes. A Pareto cover is a cover that is associated with an efficient solution of the MOSCP.

The proposed algorithm is developed based on two scalarization methods for finding efficient solutions of multiobjective optimization problems (MOPs): the weighted-sum method and the weighted-Chebyshev method, which are now briefly reviewed.

The weighted-sum method is a well-known approach to finding Pareto points located in convex regions of the Pareto frontier. The idea of this method is to convert the original MOP into a single objective optimization problem (SOP) using a convex combination of objectives. The weighted-sum function (WS) is defined as $\sum_{q=1}^{p} \lambda_q z_q(x)$, where $\lambda_q \geq 0$, $q \in Q$. The weighted-sum problem associated with the MOSCP can be written as follows:

$$\min \sum_{q=1}^{p} \lambda_q z_q(x) \qquad \text{subject to } x \in X, \tag{4.2}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_p) \in \mathbb{R}^p_{\geq}$.

When the feasible region of problem (4.2) is given as $\tilde{X} = \{x \in \mathbb{R}^n : \sum_{j \in J} a_{ij}x_j \geq 1$ for $i \in I$ and $0 \leq x_j \leq 1$ for $j \in J\}$, problem (4.2) is referred to as the weighted-sum problem of the relaxed MOSCP.

The weighted-Chebyshev method is also a well-known approach to finding Pareto points located in convex and nonconvex regions of the Pareto frontier. The weighted-Chebyshev function (WCh) with respect to the origin in $\mathcal{R}^p$ is defined as $\max_{q \in Q}\{\lambda_q(z_q(x)\}$, where $\lambda_q \geq 0, q \in Q$. This function measures the maximum deviating objective function value with respect to the origin.The weighted-Chebyshev problem associated with the MOSCP can be written as follows:

$$\begin{aligned} &\min \max_{q \in Q}\{\lambda_q z_q(x)\} \\ &\text{subject to } x \in X, \end{aligned} \tag{4.3}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_p) \in \mathbb{R}^p_{\geq}$.

4.3 Preliminary concepts

In this section we present the preliminary concepts of the proposed algorithm. The fundamental idea of the algorithm is based on the following observations.

When selecting a set with the aim of obtaining a minimum cost cover, we need to evaluate the cost of the items covered by the set and also the cost of the uncovered items. For example, let $E = \{e_1, e_2, e_3, e_4\}$ be a set of items and let $S_1 = \{e_1, e_2\}$, $S_2 = \{e_2, e_3\}$, $S_3 = \{e_3, e_4\}$ be three sets with costs $c_1^1 = 3, c_2^1 = 7, c_3^1 = 5$, respectively. If the set S_1 is selected first, the cost to cover the items e_1 and e_2 is 3 and the cost to cover the uncovered items, e_3 and e_4, is 5. Thus the total cost of the cover $\{S_1, S_3\}$ is 8. Instead, if the set S_2 is selected first, the cost to cover the items e_2 and e_3 is 7 and the cost to cover the uncovered items, e_1 and e_4, is 8. Thus the total cost of the cover $\{S_1, S_2, S_3\}$ is 15. Therefore, when selecting a minimum cost cover, it is reasonable to consider the cost of the items covered by the set and the cost to cover the uncovered items.

We also observe that when selecting a set to be in a minimum cost cover, we need to consider not only the cost of the set but also the coverage of the set. That is, it is reasonable to choose the set having a small cost and a large coverage, or equivalently, the set having a small cost to coverage ratio. For example, the cost to coverage ratio of the set S_1 is $3/2$ and that ratio of the set S_2 is $7/3$ and it is clear that selecting the set S_1 is more beneficial than selecting the S_2 since the set S_1 has the smaller ratio of $3/2$.

Based on these observation, in the construction of the algorithm, when selecting a set we consider these two aspects: the cost of uncovered items and the small cost to coverage ratio.

We first find the best set to cover each item with respect to each objective function as follows. Let D be an $m \times p$ matrix and its element $D_{i,q}$ denote the index of the best set that can be used to cover the item e_i with respect to the objective function q. The element $D_{i,q}$ is defined as:

$$D_{i,q} = \arg\min_{S_j : e_i \in S_j} \{ \tfrac{c_j^q}{|S_j|} \} \quad i \in I, q \in Q, \tag{4.4}$$

where $|S_j|$ denotes the cardinality of the set S_j.

For example, refer to the sets S_1, S_2, and S_3 defined above and assume that each has two costs: the costs $c_j^1, j = 1, 2, 3$, given above and the costs $c_1^2 = 4, c_2^2 = 3, c_3^2 = 6$.

The item e_1 is only covered by the set S_1 and therefore $D_{1,1} = 1$ and $D_{1,2} = 1$. The item e_2 is covered by the sets S_1 and S_2. Since each set contains two items, according to the ratio given in (4.4), $D_{2,1} = \arg\min\{3/2, 7/2\} = 1$ and $D_{2,2} = \arg\min\{4/2, 3/2\} = 2$. That is, the best sets to cover the item e_2 are the sets S_1 and S_2 with respect to objective function 1 and 2, respectively. By continuing this procedure, the best sets to cover each item with respect to each objective function are found and recored in the matrix D given as

$$D = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 3 & 2 \\ 3 & 3 \end{pmatrix}. \tag{4.5}$$

Based on these observations we define the estimated cost, $Est(c_j^q)$, for each set S_j with respect to the objective function q as follows:

$$Est(c_j^q) = c_j^q + \sum_{e_i \in E \setminus S_j} c_{D_{i,q}}^q, j \in J, q \in Q. \tag{4.6}$$

The estimated cost is calculated as the sum of the nominal cost of the set S_j with respect to the objective function q and the sum of the costs of the sets $S_{D_{i,q}}$ with respect to the objective function q, where the sets $S_{D_{i,q}}$ are the sets needed to cover all the other uncovered items, that is, items not in the set S_j.

In the summation of (4.6), we consider the cost of a set only one time because when a set is added to cover an uncovered item e_i, all the other uncovered items in the set are also covered. For example, consider the set S_1. The items e_1 and e_2 are covered by the set S_1 while the items e_3 and e_4 are not covered by this set. According to (4.5), the best set to cover the item e_3 is the set S_3 and the best set to cover the item e_4 is again the set S_3 with respect to objective function 1. By selecting the set S_3 we cover both e_3 and e_4. Thus to calculate $Est(c_1^1)$, we add the cost of the set S_3 only one time in (4.6) and obtain $Est(c_1^1) = 3 + 5 = 8$. But with respect to objective function 2, to cover the items e_3 and e_4 we need to select the sets S_2 and S_3, respectively. Thus, we obtain $Est(c_1^2) = 4+3+6 = 13$.

For the MOSCP, we propose two concepts for defining the estimated cost of a set. Based on the weighted-sum function, the estimated cost of the set S_j, denoted as $EstW(S_j)$, is defined as follows:

$$EstW(S_j) = \sum_{q=1}^{p} \lambda_q Est(c_j^q) \text{ where } \lambda = (\lambda_1, \ldots, \lambda_p) \in \mathcal{R}_{\geq}^p. \tag{4.7}$$

Based on the weighted Chebyshev function, the estimated cost of the set S_j, denoted as $EstC(S_j)$, is defined as follows:

$$EstC(S_j) = \max_{q \in Q}\{\lambda_q Est(c_j^q)\} \text{ where } \lambda = (\lambda_1, \ldots, \lambda_p) \in \mathcal{R}^p_{\geq}. \tag{4.8}$$

For example, let $\lambda = (0.5, 0.5)$. Continuing the example, we obtain $EstW(S_1) = 0.5(8) + 0.5(13) = 10.5$ and $EstC(S_1) = \max\{0.5(8), 0.5(13)\} = 6.5$.

4.4 Algorithm

The concepts of the cost of uncovered items and the small cost to coverage ratio lead to the development of the proposed algorithm. The algorithm consists of two phases. In the first phase the algorithm uses these concepts to search for the feasible solutions of the MOSCP while the second phase consists of improving the solutions returned by the first phase. The proposed algorithm is called the Add-Improve Algorithm (AIA). The pseudo-code of the proposed algorithm is given in Algorithm 1.

Let $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$ be the set of different $\lambda \geq \underline{0}$ vectors defined by the decision maker, A be an $m \times n$ matrix with $0-1$ coefficients a_{ij}, and C be an $p \times n$ matrix with coefficients c_j^q for $i \in I, j \in J$ and $q \in Q$. The input parameters for the algorithm are the set Λ and the matrices A and C. The algorithm generates a set of feasible solutions (denoted by $\hat{X}$) and the set of corresponding outcomes (denoted by $\hat{Y}$) of the MOSCP. At the beginning of the algorithm, these sets are initialized as empty sets, and Procedure 1 (*Dmatrix*) is called to obtain the matrix D. The pseudo-code of *Dmatrix* is given Procedure 1.

The input parameters for Procedure 1 are the matrices A and C. The parameters α_q and min_{j_q} record the minimum ratio for each item e_i and the index of the corresponding set with respect to the objective function q, respectively. For each item e_i and objective function q, the procedure initializes $\alpha_q = \infty$ and finds the best value for α_q based on (4.4). Finally, the procedure returns the matrix D.

In the algorithm, the initial input solution, the weighted-sum function, and the weighted-Chebyshev function are denoted by x, WS, and WCh, respectively. For each $\lambda \in \Lambda$, in Phase 1 three different initial solutions denoted as x^1, x^2, x^3 using Procedure 2 (*Add*) and Procedure 3 (*LpRelax*) are obtained. These solutions are improved in Phase 2 using Procedure 4 (*ColReduce*) and the improved solutions are denoted as $\hat{x}^1, \hat{x}^2, \hat{x}^3$. The pseudo-codes for *Add*, *LpRelax* and *ColReduce* procedures are givenn in Procedure 2, Procedure 3, and Procedure 4, respectively.

Algorithm 1 *Add-Improve* algorithm for the MOSCP

```
1: Input: $\Lambda$, $A$, $C$
2: Initialization: $\hat{X} = \emptyset$, $\hat{Y} = \emptyset$

3: $D \leftarrow$ Dmatrix $(A, C)$            ▷ Call Procedure 2 to obtain the matrix $D$
4: for $\lambda \in \Lambda$ do                ▷ Call Phase 1 and Phase 2 for each $\lambda \in \Lambda$
5:     Begin {Phase 1}
   ▷ Generate solutions using Procedure 2 with functions $WS, WCh$ and, Procedure 3
6:     $x^1 \leftarrow$ Add$(A, C, D, \lambda, x, WS)$
7:     $x^2 \leftarrow$ Add$(A, C, D, \lambda, x, WCh)$
8:     $x^3 \leftarrow$ LpRelax $(A, C, D, \lambda)$
9:     End {Phase 1}

10:    Begin {Phase 2}
   ▷ Improve initial solutions using Procedure 4
11:    $\hat{x}^1 \leftarrow$ ColReduce$(x^1, A, C, \lambda, WS)$
12:    $\hat{x}^2 \leftarrow$ ColReduce$(x^2, A, C, \lambda, WCh)$
13:    $\hat{x}^3 \leftarrow$ ColReduce$(x^3, A, C, \lambda, WS)$
14:    $\hat{X} \leftarrow \hat{X} \cup \{x^1, x^2, x^3\}$, $\hat{Y} \leftarrow \hat{Y} \cup \{z(\hat{x}^1), z(\hat{x}^1), z(\hat{x}^3)\}$
15:    End {Phase 2}
16: end for

17: Return: $\hat{X}$, $\hat{Y}$
```

Pseudo-code of the *Add-Improve* Algorithm

4.4.1 First phase of the algorithm

In the first phase, a set of initial feasible solutions of the MOSCP is found by using both Procedure 2 and Procedure 3. The underlying concept of these procedures is to construct the feasible solutions based on the information derived from (4.6), (4.7) and

Procedure 1 *Dmatrix*

```
1: Input: $A$, $C$

2: for $i = 1 \to m$ do
3:     for $q = 1 \to p$ do
4:         $\alpha_q = \infty$
5:         for $j = 1 \to n$ do
6:             if $e_i \in S_j$ and $\frac{c_j^q}{|S_j|} < \alpha_q$ then        ▷ Find the best set to cover item $e_i$
7:                 $\alpha_q = \frac{c_j^q}{|S_j|}$
8:                 $min_{j_q} = j$
9:             end if
10:        end for
11:        $D_{i,q} = min_{j_q}$
12:    end for
13: end for

14: Return: $D$
```

Pseudo-code of generating matrix D

(4.8).

Procedure 2 makes use of the functions *WS* or *WCh* and an infeasible solution x to construct a feasible solution. The input parameters for the procedure are the matrices A, C, D, a vector λ, the infeasible solution x, and the functions WS or WCh. The symbols $\bar{E}$, $\bar{J}$ and $InitialD$ denote the set of currently covered items, the index set of selected sets for covering the items in E, and a copy of the matrix D, respectively.

In the procedure, an unselected set S_j in the infeasible solution x is first considered (line 5 in Procedure 2). If the item e_i is not in the set S_j (i.e., $a_{ij} = 0$) and the item e_i is not covered by any other set (i.e., $e_i \in E \setminus \bar{E}$), the estimated cost of the set S_j with respect to each objective function q is evaluated based on (4.6) (line 10 in Procedure 2). The estimated nominal cost of the set S_j is then evaluated based on (4.7) if the type of the function is *WS* (line 20 in Procedure 2) or based on (4.8) if that is *WCh* (line 23 in Procedure 2). Finally, the set S_{j^*} having the minimum cost is selected (lines 29–33 in Procedure 2) as the best set to be added to the current solution x. The index j^* is added to the index set $\bar{J}$ and all items in the set S_{j^*} are added to the set $\bar{E}$. The procedure is run until all items in E have been covered. Upon termination, the procedure yields a feasible solution x.

The *LpRelax* procedure is developed based on the weighted-sum scalarization technique. The input parameters for the procedure are the matrices A, C, D and a vector λ. Using the weight vector λ, the weighted-sum problem of the relaxed MOSCP is first solved to yield a solution x (line 3 in Procedure 3). In general, the relaxed problems do not provide feasible integer solutions to the original problems. To obtain integrality, x is rounded to the nearest integer solution (lines 5–6 in Procedure 3). If the rounded x is feasible for the original MOSCP, then the feasible solution is returned. Otherwise, the procedure calls the *Add* procedure along with the infeasible solution x and the function WS to obtain a feasible solution (line 13 in Procedure 3). In any case, upon termination, the procedure returns the feasible solution x.

Procedure 2 *Add*

```
1: Input: $A, C, D, \lambda, x, function\ type = WS\ or\ WCh$
2: Initialization: $InitialD = D$

3: while $\bar{E} \neq E$ do    ▷
4:     for $j = 1 \rightarrow n$ do
5:         if $x_j = 0$ then    ▷ Select an unselected set $S_j$ in the solution $x$
6:             $D = InitialD$    ▷ Reset the matrix $D$
7:             for $i = 1 \rightarrow m$ do
8:                 for $q = 1 \rightarrow p$ do
   ▷ Select the items not in $S_j$ and not covered
9:                     if $a_{ij} = 0$ and $e_i \in E \setminus \bar{E}$ and $D_{i,q} \neq \infty$ then
   ▷ Estimate the cost of $S_j$ with respect to objective function $q$
10:                        $Est(c_j^q) = Est(c_j^q) + c^q_{D_{i,q}}$
11:                        for $k = 1 \rightarrow m$ do    ▷ Update the matrix $D$
12:                            if $D_{k,q} = D_{i,q}$ then
13:                                $D_{k,q} = \infty$
14:                            end if
15:                        end for
16:                    end if
17:                end for
18:            end for
19:            if $function\ type = WS$ then
20:                $EstW(S_j) = \sum_{q=1}^{p} \lambda_q Est(c_j^q)$
21:            end if
22:            if $function\ type = WCh$ then
23:                $EstC(S_j) = \max_{q \in Q}\{\lambda_q Est(c_j^q)\}$
24:            end if
25:        else
26:            $EstW(S_j) = \infty$ and $EstC(S_j) = \infty$    ▷ Set the estimated cost of selected sets to $\infty$
27:        end if
28:    end for
29:    if $function\ type = WS$ then
30:        $j^* = \arg\min_{j \in J}\{EstW(S_j)\}$    ▷ Select $S_{j^*}$ having the smallest cost estimation for $WS$
31:    else
32:        $j^* = \arg\min_{j \in J}\{EstC(S_j)\}$    ▷ Select $S_{j^*}$ having the smallest cost estimation for $WCh$
33:    end if
34:    $\bar{J} \leftarrow \bar{J} \cup \{j^*\}, \bar{E} \leftarrow \bar{E} \cup S_{j^*},\ x_{j^*} = 1$    ▷ Update $\bar{J}$, $\bar{E}$ and $x$
35: end while

36: Return: $x$
```

Pseudo-code of the *Add* procedure

Procedure 3 *LpRelax*

```
1: Input $A, C, \lambda, D$
2: Initialization $\bar{E} = \emptyset$

3: $x \leftarrow$ solve relaxed SOSCP with weight vector $\lambda$
4: for $j = 1 \rightarrow n$ do
5:     if $x_j \geq 0.5$ then                    ▷ Round the fractional solution
6:         $x_j = 1$
7:         $\bar{E} = \bar{E} \cup S_j$
8:     else
9:         $x_j = 0$
10:    end if
11: end for
12: if $\bar{E} \neq E$ then                      ▷ Call the Add procedure for infeasible solutions
13:     $x^w \leftarrow Add(A, C, D, \lambda, x, WS)$
14:     $x = x^w$
15: end if

16: Return $x$
```

Pseudo-code of the *LpRelax* procedure

4.4.2 Second phase of the algorithm

In the second phase the same scalarization techniques discussed above are utilized to improve the initial solutions provided by Phase 1. The underlying concept is to remove redundant sets that were included in the initial solutions. A set S_j is said to be redundant if feasibility of the current solution is guaranteed after removing the set from the solution. The improvement process occurs in the *ColReduce* procedure by eliminating redundant sets of a solution, if any are found. In this algorithm, a scalar cost is used to identify redundant sets.

For a given weight vector λ, the scalar costs of the set S_j, denoted by $C(S_j)$, are defined in (4.9) and (4.10) based on the concepts of the weighted-sum function and the weighted-Chebyshev function, respectively.

$$C(S_j) = \sum_{q=1}^{p} \lambda_q c_q^j \tag{4.9}$$

$$C(S_j) = \max_{q \in Q} \{\lambda_q c_q^j\} \tag{4.10}$$

We now give a overview of the *ColReduce* procedure. The input parameters for the procedure are the matrices A, C, D, the vector λ, the feasible solution x and the functions WS or WCh. The symbol PR denotes the index set of redundant sets and this set is initialized as an empty set.

First, for a given solution x, the scalar cost value $C(S_j)$ associated with each redundant set is calculated based on the input function type (line 6 for function *WS* and line 9 for function *WCh* in Procedure 4) and the indices of these sets are added to the set PR. Once all redundant sets have been found, starting from the set having the highest scalar cost value, the sets are removed from the solution while the feasibility of the solution holds. The procedure returns the improved solution x.

Procedure 4 *ColReduce*

```
1: Input $A, C, \lambda, D, x, function\ type = WS$ or $WCh$
2: Initialization $PR = \emptyset$

3: for $j = 1 \to n$ do
4:     if ($x_j = 1$ and $(E \setminus S_j) = E$) then                ▷ Select redundant set $S_j$
5:         if $function\ type = WS$ then
6:             $C(j) = \sum_{q=1}^{p} \lambda_q c_j^q$                ▷ Compute the weighted cost of $S_j$
7:         end if
8:         if $function\ type = WCh$ then
9:             $C(S_j) = \max_{q \in Q}\{\lambda_q c_j^q\}$          ▷ Compute the Chebyshevcost of $S_j$
10:        end if
11:        $PR = PR \cup \{j\}$                                      ▷ Update the set $PR$
12:    end if
13: end for
14: while $PR \neq \emptyset$ do
15:    $j^* = \arg\max_{j \in PR}\{C(S_j)\}$        ▷ Remove a redundant $S_j$ having the maximum cost
16:    if $E \setminus S_j = E$ then
17:        $x_j = 0$
18:    end if
19:    $PR = PR \setminus \{j\}$
20: end while

21: Return $x$
```

Pseudo-code of the *ColReduce* procedure

4.5 Computational results

This section presents the computational results obtained using the proposed Add-Improve Algorithm (AIA). The algorithm was implemented using MATLAB interface while all experiments were carried out on a personal computer with a Pentium-IV processor and 2 GB RAM. The performance of the algorithm is tested using BOSCPs generated by Gandibleux ([1]) and MOSCPs with three objective functions. The test instances generated by Gandibleux are divided into four different types A, B, C, and D based on the objective functions. To generate the cost coefficients of MOSCPs with three objective functions, we used a random number generator to produce uniform random integers between 1 and 100. In addition, to generate each coefficient a_{ij} of the matrix A, first we used a random number generator to produce uniform random numbers between zero and one. Then, if the random number generated for a_{ij} is greater than 0.5 we assign a_{ij} to be one and otherwise we assign a_{ij} to be zero.

The characteristics of all test problems are given in Table 4.1. In the left column the names of the test instances are listed while the number of objectives (p) is listed in the middle column. The number of items (m) and subsets (n) are given in the right column in this table.

Several quality measures are available in the literature ([11], [43], [53], [112]) for evaluating the performances of algorithms which are developed to obtain solutions to MOPs. In order to compare the quality of the approximations generated by the AIA with the PMA proposed by Jaszkiewicz ([53]), the *Chebyshev-scalarization measure* ($\mathcal{C}$ measure) proposed in [43] is used. In addition, the *hyper-volume measure* ($\mathcal{H}$ measure), one of the most frequently applied measures for comparing the results of multiobjective optimization algorithms, proposed in [112] is used. Let $\mathcal{C}(S)$ and $\mathcal{H}(S)$ denote the measures of a generic approximation set S with respect to the $\mathcal{C}$ and $\mathcal{H}$ measures, respectively. Let $Y_P, \tilde{Y}$ and $\hat{Y}$ denote the Pareto set, the set of outcomes obtained by solving the relaxed SCP, and the

Test problems	p	$m \times n$
2scp11A, 2scp11B, 2scp11C, 2scp11D	2	10×100
2scp41A, 2scp41B, 2scp41C, 2scp41D	2	40×200
2scp43A, 2scp43B, 2scp43C, 2scp43D	2	40×200
2scp42A, 2scp42B, 2scp42C, 2scp42D	2	40×400
2scp61A, 2scp61B, 2scp61C, 2scp61D	2	60×600
2scp62A, 2scp62B, 2scp62C, 2scp62D	2	60×600
2scp81A, 2scp81B, 2scp81C, 2scp81D	2	80×800
2scp82A, 2scp82B, 2scp82C, 2scp82D	2	80×800
2scp101A, 2scp101B, 2scp101C, 2scp101D	2	100×1000
2scp102A, 2scp102B, 2scp102C, 2scp102D	2	100×1000
2scp201A, 2scp201B, 2scp201C, 2scp201D	2	200×1000
3scp40	3	40×200
3scp60	3	60×600
3scp80	3	80×800
3scp100	3	100×1000
3scp200	3	200×1000

Table 4.1: Characteristics of the test instances

set of outcomes yield by the AIA, respectively.

In order to calculate the $\mathcal{C}$ measure of a set S given a set of weight vectors, the Chebyshev distances from the points in S to the ideal point are calculated. Then, the average of these distances over the number of weight vectors is taken as the $\mathcal{C}$ measure. For two sets S_1 and S_2, the set S_1 is better than the set S_2 with respect to this measure if $\mathcal{C}(S_1) < \mathcal{C}(S_2)$. In other words, this measure has to be minimized and its smaller value is always better. Thus, $\mathcal{C}(Y_P)$ and $\mathcal{C}(\tilde{Y})$ always provide lower bounds for $\mathcal{C}(\hat{Y})$.

Jaszkiewicz [53] uses the test problems generated by Gandibleux ([1]) and the $\mathcal{C}$ measure for evaluating the performance of the PMA. We note that for some test instances among those generated by Gandibleux our $\mathcal{C}(Y_P)$ and $\mathcal{C}(\tilde{Y})$ measures are different from those obtained by [53]. We believe that this difference is due to the ideal point selected in the calculation of these measures. Because of these differences, to better compare the performance of the proposed AIA against the performance of the PMA, two ratios R_1 and R_2 with respect to the AIA and PMA, respectively, are proposed.

Let $\mathcal{C}(Y_P)$, $\mathcal{C}(\tilde{Y})$ and $\mathcal{C}(\hat{Y})$ denote the $\mathcal{C}$ measures of the sets Y_P, $\tilde{Y}$ and $\hat{Y}$, respectively, and that are obtained with the AIA and let $\mathcal{C}(JY_P)$, $\mathcal{C}(J\tilde{Y})$ and $\mathcal{C}(J\hat{Y})$ denote the $\mathcal{C}$ measures of the same sets and obtained with the PMA as reported in [53]. In Table 4.2, $R_1 = \mathcal{C}(\hat{Y})/\mathcal{C}(Y_P)$ and $R_2 = \mathcal{C}(J\hat{Y})/\mathcal{C}(JY_P)$ while in Table 4.3, $R_1 = \mathcal{C}(\hat{Y})/\mathcal{C}(\tilde{Y})$ and $R_2 = \mathcal{C}(J\hat{Y})/\mathcal{C}(J\tilde{Y})$. Clearly, a value of each ratio close to 1.0 indicates a good approximation of the Pareto set. In addition, a smaller ratio corresponds to a better algorithm when comparing two algorithms. Tables 4.2 and 4.3 show the $\mathcal{C}$ measures and the corresponding ratios R_1, R_2 for the test problems reported in column 1.

For example, in Table 4.2, both the $\mathcal{C}(Y_P)$ and the $\mathcal{C}(JY_P)$ measures for the 2scp41A test problem is 0.1051 and the ratios R_1, R_2 are 1.0314 and 1.0438, respectively. Thus, for this test problem, the AIA performs better compared to the PMA. In both Tables 4.2 and 4.3, for each test problem, the better ratio is marked with the "*" sign which indicates the better performing algorithm. As seen in Tables 4.2 and 4.3, the AIA performs better than PMA for all test problems except for the test problems 2scp81B, 2scp201A, and 2scp201B.

	AIA			PMA		
	$\mathcal{C}(Y_P)$	$\mathcal{C}(\hat{Y})$	R_1	$\mathcal{C}(JY_P)$	$\mathcal{C}(J\hat{Y})$	R_2
2scp41A	0.1051	0.1084	1.0314*	0.1051	0.1097	1.0438
2scp41B	0.0779	0.0813	1.0436*	0.0779	0.0821	1.0539
2scp41C	0.1254	0.1468	1.1706*	0.1073	0.1269	1.1827
2scp41D	0.0734	0.0816	1.1117*	0.0668	0.0888	1.3293

Table 4.2: Comparison of the algorithms using $\mathcal{C}(Y_P)$

In Table 4.4, we report the $\mathcal{C}(Y_P)$ and $\mathcal{C}(\hat{Y})$ measures for the test problems generated by Gandibleux ([1]). For all those test problems, we were able to generate the Pareto set using the ϵ-constraint method ([29]). As seen in Table 4.4, for many test problems the ratios are close to 1 and so the AIA performs well.

The $\mathcal{H}$ measure gives the volume of the portion of the objective space dominated by the Pareto set from below and bounded by the nadir point from above. Considering two sets S_1 and S_2, $\mathcal{H}(S_1) > \mathcal{H}(S_2)$ implies that the set S_1 is better than the set S_2 based on the $\mathcal{H}$ measure. That is, this measure has to be maximized and the higher value is always

	AIA			PMA		
	$\mathcal{C}(\tilde{Y})$	$\mathcal{C}(\hat{Y})$	R_1	$\mathcal{C}(J\tilde{Y})$	$\mathcal{C}(J\hat{Y})$	R_2
2scp81A	0.0838	0.0842	1.0048*	0.0800	0.0844	1.0550
2scp81B	0.0816	0.0876	1.0735	0.0771	0.0815	1.0571*
2scp81C	0.0299	0.0544	1.8194*	0.0101	0.0205	2.0297
2scp81D	0.0168	0.0245	1.4583*	0.0137	0.0400	2.9197
2scp201A	0.0611	0.0941	1.5402	0.0590	0.0813	1.3780*
2scp201B	0.0612	0.0940	1.5359	0.0513	0.0659	1.2846*
2scp201C	0.0573	0.1602	2.7958*	0.0552	0.1649	2.9873
2scp201D	0.0837	0.4420	5.2808*	0.0789	0.4967	6.2953

Table 4.3: Comparison of the algorithms using $\mathcal{C}(\tilde{Y})$

better. Thus, $\mathcal{H}(Y_p)$ and $\mathcal{H}(\tilde{Y})$ always provide an upper bound on $\mathcal{H}(\hat{Y})$. If all outcomes produced by an algorithm are not in the region bounded by the Pareto set and the nadir point, then the $\mathcal{H}$ measure is assigned to zero.

Table 4.5 shows the $\mathcal{H}(Y_P)$ and $\mathcal{H}(\hat{Y})$ measures while Tables 6 shows the $\mathcal{H}(\tilde{Y})$ and $\mathcal{H}(\hat{Y})$ measures for the test problems generated by Gandibleux ([1]). In Table 4.6, for the 2scp201C and 2scp201D problems, we use the $\mathcal{H}(\tilde{Y})$ measure since we are not able to obtain the Pareto sets. For example, in Table 4.5, $\mathcal{H}(Y_P) = 0.7770$ for the 2scp41A test instance while $\mathcal{H}(Y) = 0.7656$. Since these values are very close to each other, the AIA outcomes are also very close to the Pareto outcomes. As seen in Table 4.5 the $\mathcal{H}(Y_P)$ and $\mathcal{H}(Y)$ measures are close to each other for many test problems except for 2scp62C. In Table 4.6, we note large differences between the $\mathcal{H}$ measures of $\tilde{Y}$ and $\hat{Y}$ for 2scp201C and 2scp201D problems. It is not clear whether this difference is mainly due to the poor performance of the AIA or due to a large gap between the sets $\tilde{Y}$ and $\hat{Y}$.

	$\mathcal{C}(Y_P)$	$\mathcal{C}(\hat{Y})$	R_1		$\mathcal{C}(Y_P)$	$\mathcal{C}(\hat{Y})$	R_1
2scp11A	0.0910	0.0964	1.0593	2scp81A	0.0808	0.0850	1.0520
2scp11B	0.1151	0.1239	1.0764	2scp81B	0.0804	0.0876	1.0896
2scp11C	0.1884	0.2130	1.1305	2scp81C	0.0169	0.0250	1.4793
2scp11D	0.1719	0.1772	1.0308	2scp81D	0.0179	0.0210	1.1732
2scp42A	0.0838	0.0867	1.0346	2scp82A	0.0617	0.0704	1.1410
2scp42B	0.0724	0.0747	1.0318	2scp82B	0.0478	0.0526	1.1004
2scp42C	0.1066	0.1159	1.0872	2scp82C	0.2713	0.3654	1.3468
2scp42D	0.1426	0.1602	1.1234	2scp82D	0.0637	0.0882	1.3846
2scp43A	0.0834	0.0905	1.0851	2scp101A	0.0451	0.0526	1.1663
2scp43B	0.0668	0.0684	1.0240	2scp101B	0.0514	0.0589	1.1459
2scp43C	0.1049	0.1128	1.0753	2scp101C	0.1688	0.2545	1.5077
2scp43D	0.1152	0.1241	1.0773	2scp101D	0.1364	0.2157	1.5814
2scp61A	0.0706	0.0805	1.1402	2scp102A	0.0433	0.0507	1.1709
2scp61B	0.0818	0.0929	1.1357	2scp102B	0.0513	0.0594	1.1579
2scp61C	0.1159	0.1551	1.3382	2scp102C	0.1178	0.1516	1.2869
2scp61D	0.0657	0.0933	1.4201	2scp102D	0.1520	0.2106	1.3855
2scp62A	0.0807	0.0808	1.0012	2scp201A	0.0700	0.0878	1.2543
2scp62B	0.0611	0.0703	1.1506	2scp201B	0.0569	0.0769	1.3515
2scp62C	0.3907	0.6447	1.6501				
2scp62D	0.1291	0.1580	1.2239				

Table 4.4: $\mathcal{C}$ measures

	$\mathcal{H}(Y_P)$	$\mathcal{H}(\hat{Y})$		$\mathcal{H}(Y_P)$	$\mathcal{H}(\hat{Y})$
2scp11A	0.8169	0.7988	2scp62B	0.9122	0.8874
2scp11B	0.7241	0.6912	2scp62C	0.3147	0.0000
2scp11C	0.4498	0.2993	2scp62D	0.7503	0.6424
2scp11D	0.4549	0.4244	2scp81A	0.8502	0.8333
2scp41A	0.7770	0.7656	2scp81B	0.8540	0.8303
2scp41B	0.8635	0.8504	2scp81C	0.9794	0.9537
2scp41C	0.7344	0.6461	2scp81D	0.9835	0.9559
2scp41D	0.8911	0.8608	2scp82A	0.9080	0.8802
2scp42A	0.8439	0.8312	2scp82B	0.9436	0.9311
2scp42B	0.8794	0.8714	2scp82C	0.7462	0.4739
2scp42C	0.7645	0.7283	2scp82D	0.9310	0.8320
2scp42D	0.6885	0.6369	2scp101A	0.9468	0.9267
2scp43A	0.8275	0.8123	2scp101B	0.9336	0.9161
2scp43B	0.8721	0.8599	2scp101C	0.8583	0.5739
2scp43C	0.7730	0.7350	2scp101D	0.8448	0.5715
2scp43D	0.7212	0.6860	2scp102A	0.9490	0.9324
2scp61A	0.8795	0.8436	2scp102B	0.9347	0.9157
2scp61B	0.8521	0.8188	2scp102C	0.7532	0.5307
2scp61C	0.7383	0.5487	2scp102D	0.6143	0.2827
2scp61D	0.8851	0.7872	2scp201A	0.8869	0.8268
2scp62A	0.8133	0.8352	2scp201B	0.9073	0.8155

Table 4.5: $\mathcal{H}(Y_P)$ and $\mathcal{H}(\hat{Y})$ measures

	$\mathcal{H}(\tilde{Y})$	$\mathcal{H}(\hat{Y})$
2scp201C	0.9134	0.3769
2scp201D	0.8467	0.0000

Table 4.6: $\mathcal{H}(\tilde{Y})$ and $\mathcal{H}(\hat{Y})$ measures

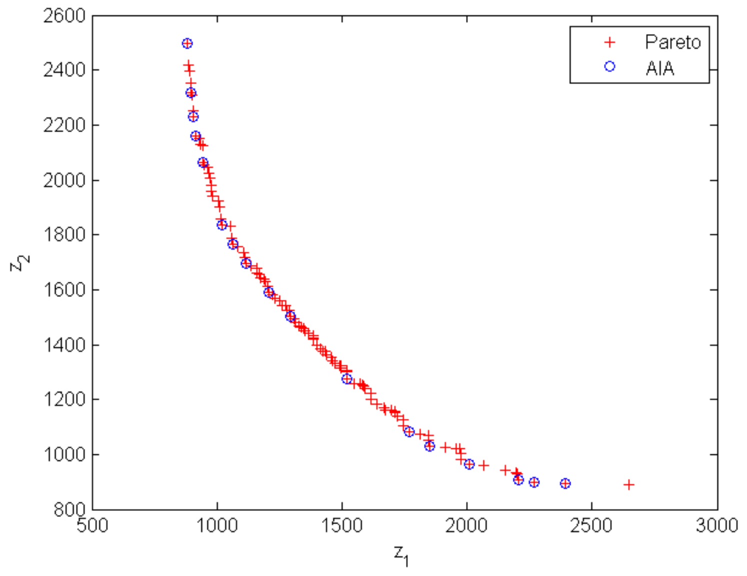


Figure 4.1: Comparison of outcomes obtained by the AIA with the Pareto outcomes for 2scp41A

Figures 4.1, 4.2, 4.3, 4.4 and 4.5 depict the Pareto outcomes and the outcomes obtained with the AIA for the test problems 2scp41A, 2scp41B, 2scp41C, 2scp41D, 2scp201B and 2scp41C, respectively. As seen in these figures, the Pareto outcomes and the outcomes obtained with the AIA are very close to each other. In Figures 4.3 and 4.4, we observe that the lower envelopes obtained by connecting adjacent Pareto points have non-convex shapes for the 2scp41C and 2scp41D test problems. Despite the nonconvexity, the AIA obtains outcomes located in the non-convex regions. Also, as seen in Figure 4.5, the AIA does not find any Pareto outcomes for the 2scp201B test problem, but rather obtains outcomes that are very close to the Pareto outcomes. Figure 4.6 depicts the points in the set $\tilde{Y}$ and the AIA outcomes for the 2scp201C test problem. We note that there is a very large gap between the points in $\tilde{Y}$ and the AIA outcomes. Again, it is not clear whether this gap is mainly due to the poor performance of the AIA on this test problem or due to the gap between the sets $\hat{Y}$ and $\tilde{Y}$.

In Table 4.7, we report the $\mathcal{C}(\tilde{Y})$, $\mathcal{C}(\hat{Y})$, $\mathcal{H}(\tilde{Y})$ and $\mathcal{H}(\hat{Y})$ measures for the the test problems with three objective functions. The set $\tilde{Y}$ is used since we are not able to find

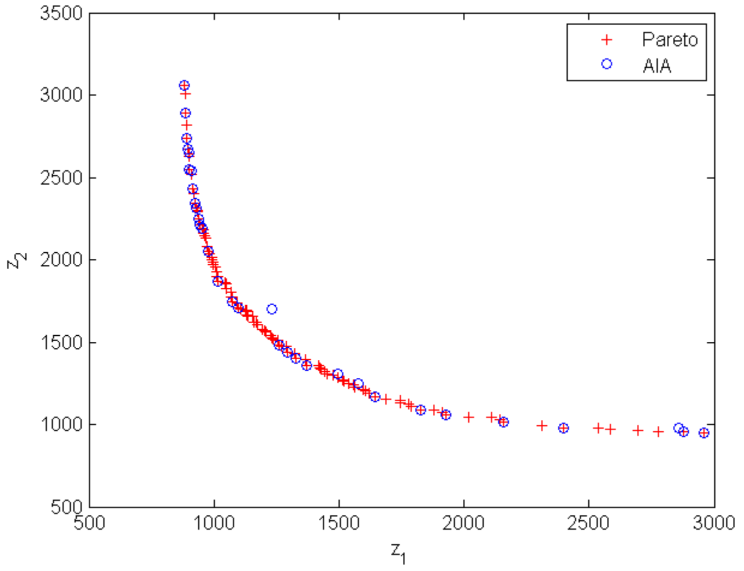


Figure 4.2: Comparison of outcomes obtained by the AIA with the Pareto outcomes for 2scp41B

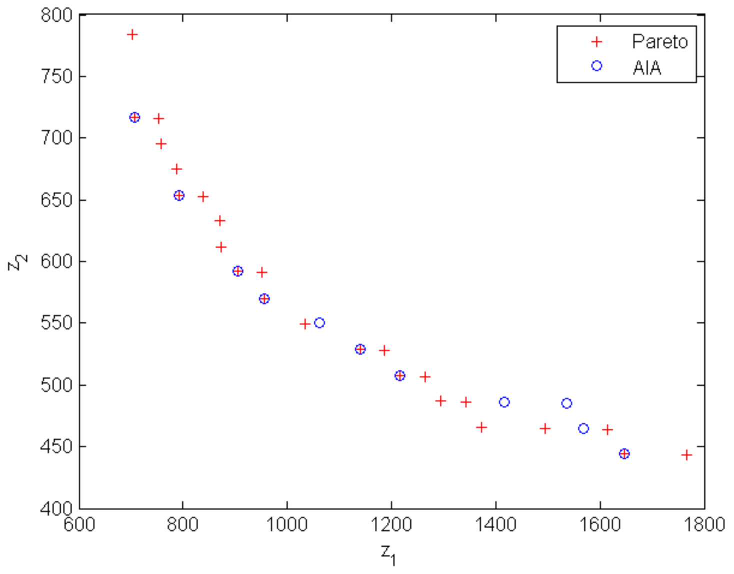


Figure 4.3: Comparison of outcomes obtained by the AIA with the Pareto outcomes for 2scp41C

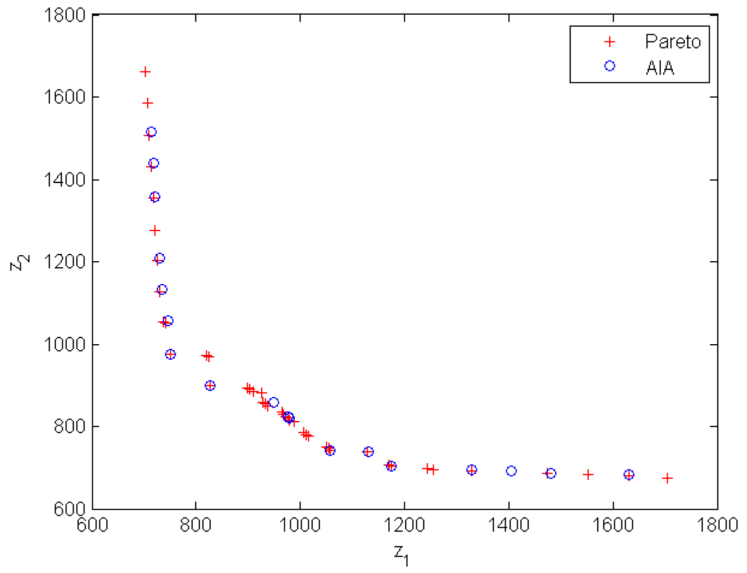


Figure 4.4: Comparison of outcomes obtained by the AIA with the Pareto outcomes for 2scp41D

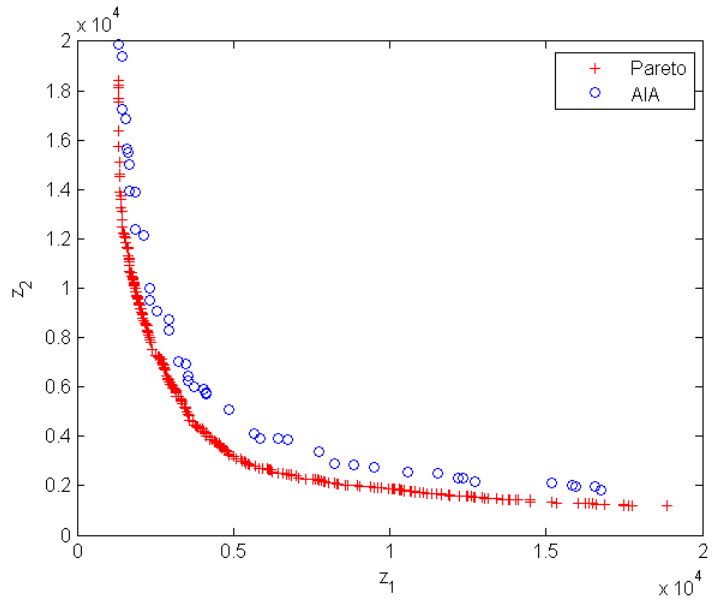


Figure 4.5: Comparison of outcomes obtained by the AIA with the Pareto outcomes for 2scp201B

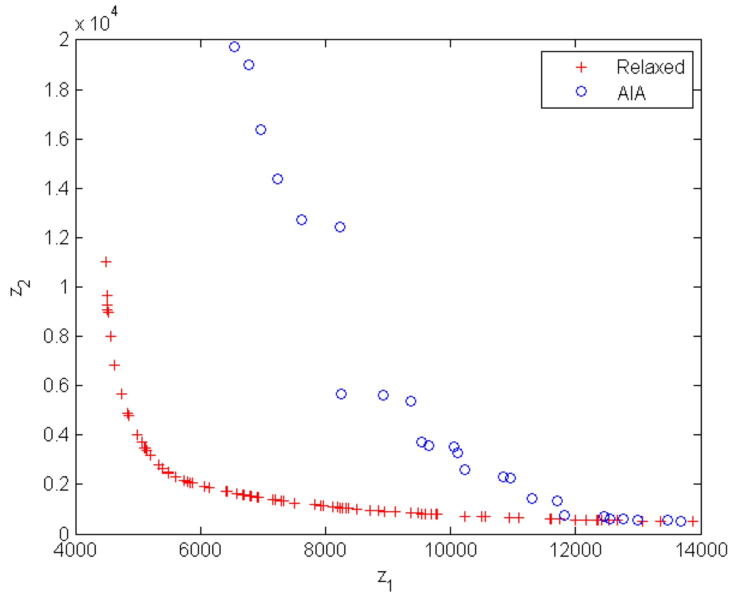


Figure 4.6: Comparison of outcomes obtained by the AIA with the relaxed outcomes for 2scp201C

	$\mathcal{C}(\tilde{Y})$	$\mathcal{C}(\hat{Y})$	$\mathcal{H}(\tilde{Y})$	$\mathcal{H}(\hat{Y})$
3scp40	0.0093	0.0128	0.5876	0.3448
3scp60	0.3836	0.5124	0.7386	0.6764
3scp80	0.2297	0.2978	0.6807	0.6302
3scp100	0.0008	0.0039	0.7651	0.2602
3scp200	0.2051	0.2412	0.7285	0.0000

Table 4.7: $\mathcal{C}$ and $\mathcal{H}$ measures for MOSCP with three objective functions

Pareto sets Y_P sets for problems with three objective functions. Table 4.7 shows that both $\mathcal{C}(\tilde{Y})$ and $\mathcal{C}(\hat{Y})$ are approximately equal to each other for all test problems. Thus we conclude that the AIA performs well. In addition, based on the $\mathcal{H}$ measure, we observe that the AIA performs well on the test problems 3scp60 and 3scp80, while it provides poor performance on the test problems 3scp100 and 3scp200. However, based on these results, we conclude that the AIA performs well on MOSCPs with a higher number of objectives.

4.6 Conclusion

A two-phase heuristic algorithm, named AIA, to approximate the Pareto set of the MOSCP is proposed based on two scalarization methods. A set of initial feasible solutions is generated in the first phase and the initial solutions are improved in the second phase. Two well-known measures, ([43], [112]), are used to evaluate the quality of approximation and the performance of the AIA is compared with the performance of the PMA ([53]). Although, it can not be decided that one heuristic algorithm is superior to another, the AIA performs better than the PMA on a majority of the test problems for which a comparison is possible. Additionally, based on other numerical and graphical results, we note that the AIA performs well on other test problems including problems with three objective functions.

Future research may lead in several directions. To improve the quality of the initial solutions, it might possible to consider other optimization schemes than the two methods considered in the first phase. In addition, more sophisticated improvement strategies can be used to improve the solutions in the second phase. Conducting additional numerical testing using MOSCPs with a higher number of objective functions will also be helpful.

Chapter 5

A Hierarchical Approach to Designing Compact Ecological Reserve Systems

5.1 Introduction

Methods to design reserve systems for ecological species have been considered by several papers in the recent literature. In general, the problem is to find a subset of reserve sites that minimizes the cost of establishing reserve sites containing a given set of species or that maximizes the number of species present under a given budget constraint. Both of these types of reserve site selection problems can be formulated as integer programming (IP) problems and represented as either a set covering problem (SCP) or a maximal covering problem (MCP).

In the set covering formulation, given a set of target species and a set of potential sites, we wish to determine the least-cost reserve system that satisfies a specified minimum representation for each species. In the maximal covering formulation, with a limited conservation budget, the objective is to determine a reserve system that includes the maximum number of species. These aspects have been considered in [75, 88] as well as in other papers.

In practice, reserve design problems need to consider more than just species coverage and budget limitation [66]. Other spatial characteristics such as the distance between selected reserve sites and the shape of the reserve system should be considered as well. Several mathematical models that consider spatial optimization have been proposed to address the important issues of representing species within compact reserve systems [68, 71, 77, 105]. Such approaches make it possible to design a better spatial arrangement for a reserve system by considering attributes such as contiguity and the shape of the selected sites.

Because the SCP and MCP formulations do not explicitly consider spatial relationships between the sites selected for the reserve system, the resulting reserve system may be highly fragmented. This may be desirable in certain cases, but more generally species can disappear quickly from isolated fragments through well-known processes: chance variations in births, death and sex of individuals (demographic stochasticity), the random loss of genetic variation (genetic drift), inbreeding, disease, and the like [35, 84]. Fragmented populations are also less able to move or evolve to survive changing conditions, and all these threats are greater in rare species, precisely the ones we wish to protect with reserves. In addition, a contiguous reserve system helps species to roam freely within the system in the face of fluctuations in density, threats (e.g., predators, fires), and environmental conditions (e.g., drought, climate change) without leaving the space. This will add to species resilience and persistence.

More compact reserve systems also help to reduce the edge effects of the system, which are generally negative [107]. For the same reserve area, a longer boundary will cost more to fence and patrol. A longer boundary will also allow more protected animals to leave the reserve, and will allow more threats from the outside to enter. Well-known threats associated with park boundaries include poaching, wood cutting, grazing, invasive and exotic species, disease, predation by feral cats and dogs, and altered climate from neighboring deforested lands. A variety of shape measures have been proposed in reserve selection models to represent compactness. Some of the proposed measures are the boundary length of the reserve, the ratio of boundary length to area, and the average distance between sites in the reserve system. A review of such measures and their importance for the reserve design problem are discussed in detail by Williams et al. [106].

Therefore contiguity and compactness can be important in realistically modeling reserve site selection problems. A variety of formulations have been proposed to address these contiguity and compactness attributes. Some formulations have explicitly considered contiguity and compactness together (using a linear combination) while others have considered only one attribute. We now briefly discuss some of these studies.

Shirabe [98] proposed an exact formulation for structural contiguity that can be incorporated into any mixed integer programming model. According to this model, the resulting system enforces contiguity regardless of the other included criteria such as compactness. Xiao et al. [109] proposed an evolutionary algorithm to maximize the relative contiguity in reserve network design. Onal and Briers [78] developed a linear IP formulation that uses a graph theory perspective to obtain a connected reserve system. Although this formulation ensures contiguity, it contains some "gap" sites that are to be excluded from the final solution. Therefore their objective was to minimize the total number of gap sites. They used additional variables and constraints to avoid cycle formation. Onal and Wang [79] developed an improved linear IP formulation, also using a graph theory approach. Their objective was again to minimize the total number of gap sites. The main difference between these two formulation is the method used to avoid cycle formation. Although the model in [78] explicitly uses additional constraints and variables to avoid cycles, the improved model [79] does not. Rather, if cycles are present in the solution, new constraints are added and the model is solved again. The authors of the improved model [79] mentioned that their model is computationally more efficient because of its reduced size. Both of these formulations focus on the structural contiguity of a reserve system. Hof and Flather [47] developed a different nonlinear IP model that preserves the contiguity of the system by controlling the shape, requiring reserves to be either circular or rectangular. This seems unnecessarily restrictive, as distributions of populations and their habitats are unlikely to match these patterns. Wu and Murray [108] proposed an unbiased relative measure of contiguity of a reserve system ranging from zero to one based on graph theory and spatial interaction.

Several mathematical models have been proposed to group disconnected sites together into compact reserves. A collection of adjacent reserve sites defines a *cluster*. In these models [33, 34, 68, 71, 72, 76] such reserves of compact shapes are generated as clusters. In an ecological sense, clusters might correspond to larger areas of single habitats or even different habitats. This might enhance opportunities for local dispersal, and preserve species interactions across the larger landscape. On the other hand, separated clusters (habitats) may be desirable because such separation will preserve species in the face of natural disasters such as destruction of the habitat by fire.

Onal and Briers [76] developed two IP approaches to the problem of reserve selection to obtain compact reserve systems. In the first approach, they minimized the sum of distances between all pairs of sites included in the reserve system. In the second approach, an alternative formulation minimizes the largest distance between selected sites instead of the total distance. Fischer and Church [33] presented a linear IP formulation for minimization of the boundary length to promote reserve aggregation and compactness.

Fischer and Church [34] developed a bi-objective formulation by considering both the boundary length and the site selection cost. McDonnell et al. [68] developed a bi-objective nonlinear IP formulation that involves a weighted combination of the boundary length of the selected clusters and the area of selected sites. They mentioned that minimizing the area of the selected sites is equivalent to minimizing the cost of the selected sites. Nalle et al. [71, 72] developed a nonlinear formulation that explicitly addresses the compactness and shape of the selected reserve sites. This model minimizes a weighted combination of two measures: the boundary length of selected clusters and the distance between all pairs of selected sites (even those in disjoint clusters).

The current paper develops a bi-objective optimization model for clustering reserve sites into a relatively small number of compact groups. One difference between this new formulation and those given in [71, 72] is the way we measure distance between selected sites. We measure the distances within each cluster, whereas the models in [71, 72] measure the distances between all selected sites whether such sites are in the same cluster or different clusters. We argue that consideration of distance within clusters rather than the total distance between all sites may be more meaningful. Since each cluster might be treated as a different habitat, in general it is not important to consider the distance between different habitats. In fact, minimizing distances between separate clusters will tend to yield reserve clusters that are close to one another, at a cost of a reduced ability to capture variation in species that we typically see with distance. In addition, there are reduced biological interactions among geographically separated clusters. For example if one habitat represents a mountain and the other habitat represents a swamp, then the species they support may be quite different and unlikely to interact, so there would be no need to try to find some compromise site in between to protect. Therefore, there is no need to measure the distance between these two habitats to obtain compact clusters. Thus if a relatively small distance can be maintained between all sites within a given cluster, this will assure maximum interactions among different species within the cluster. Therefore we concentrate on minimizing the distance within clusters in designing compact clusters.

When creating a compact cluster, the boundary length of the cluster is especially important. For example, Figure 5.1(a) and Figure 5.1(b) illustrate two possible clusters, each containing four sites. The cluster in Figure 5.1(a) has ten boundary edges while the cluster in Figure 5.1(b) has eight boundary edges. Since each site is a unit square, the number of boundary edges is equivalent to the overall boundary length. Visually it is clear that the cluster in Figure 5.1(b) is more compact than the cluster in Figure 5.1(a). Therefore when compact clusters are desired, it seems reasonable to pick clusters having a small boundary length relative to the area. In our models we therefore treat boundary length as our primary objective.

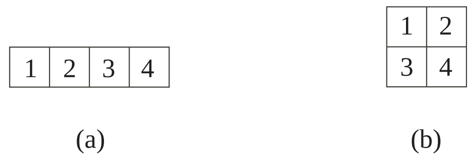


Figure 5.1: Two clusters on four sites with different boundary lengths

But simply minimizing the boundary length does not ensure compact clusters for the system. To illustrate, Figure 5.2 shows two clusters, each containing three (unit square) sites and each having the same boundary length eight. Consider now the Euclidean distance between the centers of each site in a cluster. The sum of Euclidean distances between distinct sites in Figure 5.2(a) is 4 while the corresponding sum in Figure 5.2(b) is 3.41. Visually it is clear that the cluster in Figure 5.2(b) is more compact than that in Figure 5.2(a). Therefore it seems reasonable to select among clusters with the same boundary length one having the smallest sum of (Euclidean) distances. The within cluster distance thus provides a secondary criterion. As suggested by this example, our strategy for creating a compact cluster involves two steps. We first consider the boundary length of the cluster and among all such clusters of minimum boundary length we identify a cluster that minimizes the distance between all pairs of sites within the cluster.

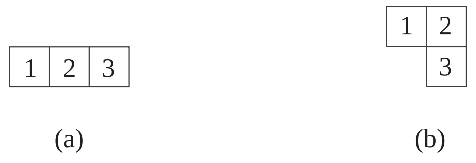


Figure 5.2: Two clusters on three sites with the same boundary length

In general we consider both aspects in designing a compact reserve system containing *several* clusters: minimizing the boundary length of all clusters and minimizing the total distances between all pairs of sites within each cluster. This should lead to a reasonably compact reserve system. Thus we formulate the reserve design problem using an appropriate hierarchical optimization model. This is another distinguishing feature of our model.

To minimize the hierarchical combination of boundary length and then total within cluster distance, we use a technique from multi-objective programming [29] that appropriately weights these two objectives. The combining weight U is specified in a particular manner to give priority to minimizing the boundary length as the primary objective. We can calculate this weight U in advance by considering the sum of l_1 (rectilinear) distances between all possible pairs of sites in the reserve.

The objective function of our optimization model is thus to minimize U times the boundary length of all clusters plus the sum of Euclidean distances between sites within the designated clusters. This formulation also incorporates the requirements to cover all target species with a limited conservation budget.

Our initial formulation is a nonlinear integer programming model and therefore solving the model exactly is time consuming. In order to solve the model more efficiently, we convert the proposed model into a linear mixed integer program. Details of this conversion are discussed in Section 5.2. In Section 5.3 we provide numerical examples to compare computational aspects of the two models discussed in Section 5.2. Additional data sets are then used to explore more fully the computational behavior of the linear mixed integer programming model as parameters of the model are varied. In Section 5.4 we apply our optimization approach to a standard data set based on Oregon field data. Section 5.5 summarizes our work and briefly outlines a heuristic approach to the design of compact reserve systems.

5.2 Optimization Models

This section presents optimization models that implement the proposed hierarchical approach for protecting certain species occurring within a given region. It is assumed that the prevalence of existing species does not change with time. Also, the entire region is considered to be partitioned into a number of potential reserve sites. Two sites are said to be adjacent if they share a common boundary.

The models developed in this section assume that the study region is partitioned into uniformly sized sites. For simplicity, the region is considered to be a rectangular $n \times m$ grid of uniform sites and each site is a 1×1 unit square. (A grid of hexagonal sites can also be easily accommodated.) Let V denote the set of sites (or nodes) in the reserve system and let E denote the set of edges (adjacencies) in the system. In the given $n \times m$ grid, any node $v \in V$ can be written as (i, j) where $i = 1, \ldots, n$ and $j = 1, \ldots, m$. If two nodes $v_1 = (i, j), v_2 = (k, l) \in V$ are adjacent (i.e., their corresponding sites share a boundary) then the ordered pair $(v_1, v_2) = ((i, j), (k, l)) \in E$. For a rectangular grid system, each site can be adjacent to at most four other sites. Other notation used in our mathematical models is described below:

S = total number of conservation species to be protected

C = maximum number of possible clusters

A_s = set of sites inhabited by species of type s where $s = 1, 2, \ldots, S$

n_s = required number of selected sites for species s where $s = 1, 2, \ldots, S$

$N(i, j) = \{(k, l) \in V : ((i, j), (k, l)) \in E\}$, the set of nodes adjacent to node (i, j)

$D(i, j) = \{(k, l) \in V : \text{either } (k \geq i \text{ and } l > j) \text{ or } (k > i \text{ and } l \leq j)\}$

$d_{ij,kl}$ = Euclidean distance between the center of site (i, j) and the center of site (k, l)

b_{ij} = budgetary cost of conserving or purchasing site $(i, j) \in V$

B = total budget available for the reserve system.

5.2.1 A nonlinear integer programming model

In this model, we define decision variables to indicate which sites are included in the reserve and their allocation to clusters:

$$X_{cij} = \begin{cases} 1 & \text{if site } (i, j) \text{ is included in Cluster } c \\ 0 & \text{otherwise.} \end{cases}$$

Here, Cluster 1 denotes the sites *not* selected for conservation and the remaining clusters contain those sites that are selected for conservation. Clusters $c = 2, 3, \ldots, C$ are called *real* clusters since they are the ones containing the protected species. Because Cluster 1 is not used for conservation purposes, we actually have at most $(C - 1)$ real clusters.

The 0-1 quadratic optimization model can be written as follows:

$$(P_1) \ \ \text{minimize} \ \ \sum_{c=2}^{C} \sum_{(i,j)\in V} \sum_{(k,l)\in D(i,j)} d_{ij,kl} X_{cij} X_{ckl} + U \times \left[\sum_{(i,j)\in V} \sum_{(k,l)\in N(i,j)} X_{1ij}(1 - X_{1kl}) \right]$$

subject to

$$\sum_{c=2}^{C} \sum_{(i,j)\in A_s} X_{cij} \geq n_s, \ \text{for all } s = 1, 2, \ldots, S \tag{5.1}$$

$$\sum_{c=2}^{C} \sum_{(i,j)\in V} b_{ij} X_{cij} \leq B \tag{5.2}$$

$$X_{c_1 ij} + X_{c_2 kl} \leq 1, \ \text{for all } ((i,j),(k,l)) \in E \text{ and } c_1 > 1, \ c_2 > 1, \ c_1 \neq c_2 \tag{5.3}$$

$$\sum_{c=1}^{C} X_{cij} = 1, \ \text{for all } (i,j) \in V \tag{5.4}$$

$$X_{cij} \in \{0, 1\}, \ \text{for all } (i,j) \in V \text{ and } c \geq 1$$

Let us consider each part of model (P_1) in detail. The objective function consists of two parts: the weighted sum of (a) the distances between sites within the same real clusters and (b) the boundary length of all real clusters. The parameter U is chosen to be sufficiently large to give priority to minimizing the boundary length.

The first summation in the objective function calculates the total Euclidean distance between sites $(i,j) \in V$ and $(k,l) \in D(i,j)$ within real clusters. Here the set $D(i,j)$ contains the distinct sites that occur “after” site (i, j) in a left-right, top-bottom ordering. The second summation gives the total number of boundary edges of all real clusters and thus the boundary length of all selected sites. We justify this claim as follows.

If both (i, j) and (k, l) are selected for Cluster 1, then $X_{1ij} = X_{1kl} = 1$ and the product $X_{1ij}(1 - X_{1kl}) = 0$ so no edge is counted towards the total boundary length. If (i, j) is in Cluster 1 and (k, l) is in a real cluster, then the product $X_{1ij}(1 - X_{1kl}) = 1$ since $X_{1kl} = 0$ and thus 1 is counted towards the total number of boundary edges. That is, since a real cluster is surrounded by Cluster 1, an edge is counted in the product above precisely when (i, j) is in Cluster 1 and (k, l) is in a real cluster. Therefore the second summation of the objective function gives the total number of boundary edges of all selected real clusters. Counting the total number of boundary edges then gives the boundary length of all clusters since we assume unit sized sites.

In the formulation of model (P_1), it is convenient to place a border of Cluster 1 sites surrounding the actual reserve system. This ensures that the outermost reserve sites are all adjacent to Cluster 1. Therefore if we have an $n \times m$ grid we are actually modeling an $(n - 2) \times (m - 2)$ reserve system.

The constraints of the model are of the following types. Constraint (5.1) is the species representation requirement which states that to protect species of type s adequately, we must select at least n_s sites in which species s is present. This general formulation allows the flexibility to protect some species – those of greater conservation value – in more reserves than others. Since only the selected reserve sites contribute to species representations, we consider only real clusters $(c > 1)$ in this constraint. Constraint (5.2) guarantees that the total cost of selected sites in the real clusters cannot exceed the conservation budget. Constraint (5.3) enforces that if sites $(i, j), (k, l)$ are included in two *different* real clusters then they do not share a boundary. This ensures that real clusters are disjoint from one another. Constraint (5.4) states that each site is assigned to exactly one cluster $c \geq 1$.

5.2.2 A linear integer programming model

The objective function for model (P_1) is a quadratic function with nmC binary variables. Unfortunately, the second summation is not a convex function of the variables X_{cij} and so model (P_1) is a nonlinear, nonconvex formulation. In order to model the problem more effectively, we convert all quadratic terms of the objective function into linear terms by replacing each quadratic term $X_{cij}X_{ckl}$ by a new binary variable Y_{cijkl} :

$$Y_{cijkl} = \begin{cases} 1 & \text{if both } (i,j),(k,l) \in V \text{ are assigned to Cluster } c \\ 0 & \text{otherwise.} \end{cases}$$

The following constraints ensure that Y_{cijkl} equals 1 if and only if sites (i,j) and (k,l) are both selected for Cluster $c \geq 1$:

$$Y_{cijkl} \leq X_{cij}, \text{ for all } (i,j),(k,l) \in V \tag{5.5}$$

$$Y_{cijkl} \leq X_{ckl}, \text{ for all } (i,j),(k,l) \in V \tag{5.6}$$

$$X_{cij} + X_{ckl} - Y_{cijkl} \leq 1, \text{ for all } (i,j),(k,l) \in V \tag{5.7}$$

$$Y_{cijkl} \geq 0, \text{ for all } (i,j),(k,l) \in V \tag{5.8}$$

Namely, constraints (5.5)–(5.6) ensure that Y_{cijkl} must equal 0 unless both X_{cij} and X_{ckl} equal 1, while constraint (5.7) ensures that if both (i,j) and (k,l) are selected for Cluster c, then Y_{cijkl} must equal 1. Thus $Y_{cijkl} = X_{cij}X_{ckl}$ always holds and the following linearized model (P_2) does not change the optimal solution of the original quadratic model (P_1):

$$(P_2) \text{ minimize } \sum_{c=2}^{C} \sum_{(i,j)\in V} \sum_{(k,l)\in D(i,j)} d_{ij,kl} Y_{cijkl} + \; U \times \left[\sum_{(i,j)\in V} \sum_{(k,l)\in N(i,j)} X_{1ij} - Y_{1ijkl} \right]$$

subject to

$$\sum_{c=2}^{C} \sum_{(i,j)\in A_s} X_{cij} \geq n_s, \text{ for all } s = 1, 2, \ldots, S \tag{5.9}$$

$$\sum_{c=2}^{C} \sum_{(i,j)\in V} b_{ij} X_{cij} \leq B \tag{5.10}$$

$$X_{c_1 ij} + X_{c_2 kl} \leq 1, \text{ for all } ((i,j),(k,l)) \in E \text{and } c_1 > 1,\ c_2 > 1,\ c_1 \neq c_2 \tag{5.11}$$

$$\sum_{c=1}^{C} X_{cij} = 1, \text{ for all } (i,j) \in V \tag{5.12}$$

$$Y_{cijkl} \leq X_{cij}, \text{ for all } (i,j) \in V,\ (k,l) \in N(i,j) \text{ and } c = 1 \tag{5.13}$$

$$Y_{cijkl} \leq X_{ckl}, \text{ for all } (i,j) \in V,\ (k,l) \in N(i,j) \text{ and } c = 1 \tag{5.14}$$

$$X_{cij} + X_{ckl} - Y_{cijkl} \leq 1, \text{ for all } (i,j) \in V,\ (k,l) \in D(i,j) \text{ and } c > 1 \tag{5.15}$$

$$X_{cij} \in \{0,1\}, \text{ for all } (i,j) \in V \text{ and } c \geq 1$$

$$Y_{cijkl} \geq 0, \text{ for all } (i,j) \in V,\ (k,l) \in N(i,j) \text{ and } c = 1$$

$$Y_{cijkl} \geq 0, \text{ for all } (i,j) \in V,\ (k,l) \in D(i,j) \text{ and } c > 1$$

This model incorporates further simplifications. Consider constraints (5.13)–(5.14). As explained earlier, $Y_{cijkl} \leq X_{cij}$, $Y_{cijkl} \leq X_{ckl}$ and $Y_{cijkl} \geq 0$ ensure that Y_{cijkl} must equal zero unless both (i,j) and (k,l) are selected. The first summation in the objective function of model (P_2) contains Y_{cijkl} for $c > 1$ with a positive weight $d_{ij,kl}$. Thus in this case the objective function forces Y_{cijkl} to be as small as feasibly possible in order to minimize the objective function value. Therefore it is unnecessary to define (5.13)–(5.14) explicitly for real clusters $c > 1$. Moreover, variables Y_{1ijkl} occur in measuring the boundary length of a real cluster, where $(i, j) \in V$ and $(k, l) \in N(i, j)$. Thus constraints (5.13)–(5.14) only need to be defined for adjacent sites when $c = 1$.

The second summation in the objective function contains $-Y_{cijkl}$ with $c = 1$. In this case the negative sign in the objective function forces all Y_{cijkl} to be as large as feasibly possible in order to achieve a minimum value. Since constraint (5.15) ensures that Y_{cijkl} is equal to one when both sites (i, j) and (k, l) are selected, it is unnecessary to define constraint (5.15) explicitly for Cluster 1. Moreover to measure the distance between two sites (i, j) and (k, l), we consider $(i, j) \in V$ and $(k, l) \in D(i, j)$. Therefore when $c > 1$, constraint (5.15) only needs to be defined for $(i, j) \in V$ and $(k, l) \in D(i, j)$.

The binary variables Y_{cijkl} are then defined for the two index sets: $\{(i, j) \in V,\ (k, l) \in D(i, j),\ c > 1\}$ and $\{(i, j) \in V,\ (k, l) \in N(i, j),\ c = 1\}$. Moreover, the nature of the objective function and constraints (5.13)–(5.15) allows us to relax the binary Y variables to be continuous variables: namely $Y_{cijkl} \geq 0$.

Model (P_2) is then a linear, mixed integer optimization model. As seen in Table 5.1, it contains a larger number of variables and constraints compared to model (P_1). But if we consider the efficiency of solving these optimization models, the second formulation is expected to be computationally preferable because it is a *linear* (mixed) integer programming model as opposed to a nonlinear, nonconvex integer model. This expectation is clearly shown by our computational experience, discussed in Section 5.3.

	model (P_1)	model (P_2)
binary variables (X)	$O(nmC)$	$O(nmC)$
continuous variables (Y)		$O(n^2m^2C)$
constraints	$O(nmC^2)$	$O(n^2m^2C)$

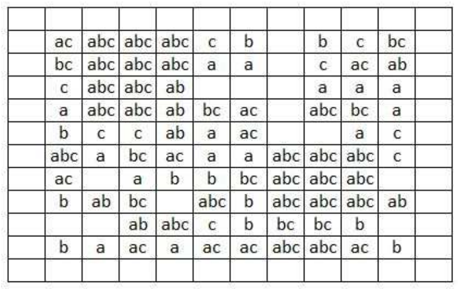
Table 5.1: Relative sizes of the two models

5.3 Numerical Results

This section presents computational results with models (P_1) and (P_2) on a variety of synthetic data sets. All experiments were carried out on a Samsung computer with a Intel Core i3-2370M processor and 4 GB RAM. Each optimization problem was formulated using OPL (Optimization Programming Language), a modeling language for Linear and Integer Programming. OPL uses the solver CPLEX, version 12.4 ([2]).

5.3.1 Comparison of models (P_1) and (P_2)

We first study models (P_1) and (P_2) on the 12×12 study region shown in Figure 5.3. This reserve system contains three species a, b, c. Those species present in each site of the grid are so indicated in Figure 5.3.



	ac	abc	abc	abc	c	b		b	c	bc	
	bc	abc	abc	abc	a	a		c	ac	ab	
	c	abc	abc	ab				a	a	a	
	a	abc	abc	ab	bc	ac		abc	bc	a	
	b	c	c	ab	a	ac			a	c	
	abc	a	bc	ac	a	a	abc	abc	abc	c	
	ac		a	b	b	bc	abc	abc	abc		
	b	ab	bc		abc	b	abc	abc	abc	ab	
			ab	abc	c	b	bc	bc	b		
	b	a	ac	a	ac	ac	abc	abc	ac	b	

Figure 5.3: 12×12 study region

Suppose that we need to conserve at least 20 sites for species of type a, 20 for species of type b, and 15 for species of type c. As seen in Figure 5.3, initially the region contains 57 potential sites for species of type a, 53 for species of type b, and 51 for species of type c. For simplicity, we assume that the budgetary cost of conserving each site is 1, so the budget constraint simply becomes an upper bound UB on the number of selected sites. The maximum number of allowable clusters C for this problem is set equal to 3.

Case 1

When UB $= 20$, the optimal solution given by both models (P_1) and (P_2) is depicted in the left portion of Figure 5.4. Since we allowed a maximum of three clusters ($C = 3$), the optimal solution contains two real clusters whose sites are denoted 2 and 3, and are shown shaded in Figure 5.4. Those sites of Cluster 1 (the non-selected sites) are designated by an empty cell. The species coverage requirements for each type are 20, 20, 15 respectively and the selected two real clusters cover 20 of type a, 20 of type b, and 20 of type c. It is clear that the two selected real clusters display ideal compact shapes.

Case 2

Here a maximum of UB $= 35$ sites can be selected and we use the same species coverage requirements. The optimal solution for this case given by both models (P_1) and (P_2) is depicted in the right portion of Figure 5.4. Although 35 sites are allowed, the optimal solution produced used only 30 sites to satisfy all stipulated species coverage requirements. Here 20 species of type a, 22 of type b, and 20 of type c are covered by the optimal solution. Note that only a single real cluster with boundary length 22 is created, denoted 2 in the right portion of Figure 5.4, in contrast to the real two clusters in the left portion of Figure 5.4 (having total boundary length 26). This reduction in boundary length occurs by increasing the number of allowable sites in the reserve system.

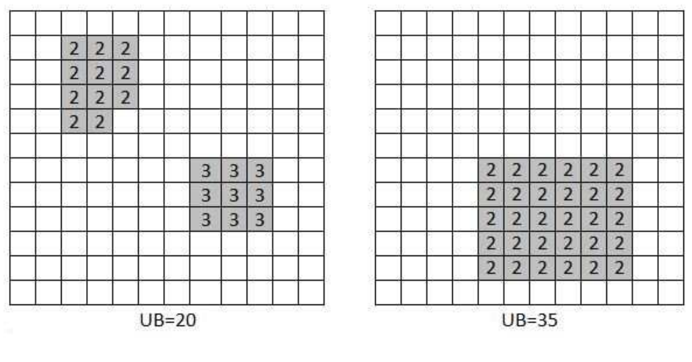
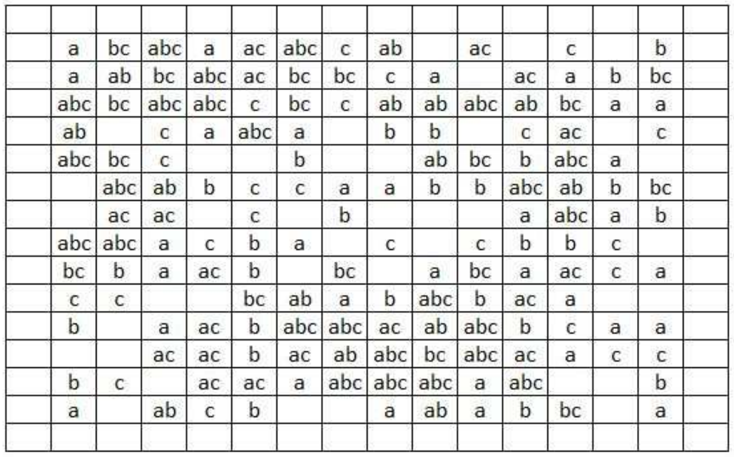


Figure 5.4: Optimal solutions for Case 1 (UB $= 20$) and Case 2 (UB $= 35$), 12×12 study region

Model (P_1) took nearly 62 seconds to provide the globally optimal solution whereas model (P_2) took only 2 seconds to provide the same optimal solution for Case 1. For Case 2, model (P_1) required 20 minutes and 45 seconds while model (P_2) took 22 seconds. This substantial reduction in computation time was observed in all test instances, so we use model (P_2) rather than model (P_1) in all our subsequent numerical investigations.

5.3.2 Varying the number of clusters

In this section we discuss how the optimal solutions behave when the allowable number of clusters C is changed. We illustrate this by considering a sample 16×16 reserve system involving species a, b, c; see Figure 5.5.



	a	bc	abc	a	ac	abc	c	ab		ac		c		b	
	a	ab	bc	abc	ac	bc	bc	c	a		ac	a	b	bc	
	abc	bc	abc	abc	c	bc	c	ab	ab	abc	ab	bc	a	a	
	ab		c	a	abc	a		b	b		c	ac		c	
	abc	bc	c			b			ab	bc	b	abc	a		
		abc	ab	b	c	c	a	a	b	b	abc	ab	b	bc	
		ac	ac		c		b				a	abc	a	b	
	abc	abc	a	c	b	a		c		c	b	b	c		
	bc	b	a	ac	b		bc		a	bc	a	ac	c	a	
	c	c			bc	ab	a	b	abc	b	ac	a			
	b		a	ac	b	abc	abc	ac	ab	abc	b	c	a	a	
			ac	ac	b	ac	ab	abc	bc	abc	ac	a	c	c	
	b	c		ac	ac	a	abc	abc	abc	a	abc			b	
	a		ab	c	b			a	ab	a	b	bc		a	

Figure 5.5: Sample conservation region with 256 sites

Model (P_2) was then solved using $C = 2$, $C = 3$, and $C = 4$. As we increase the number of allowable clusters for the 16×16 grid, the optimal boundary length BL, the optimal within cluster distance DIS, and the elapsed CPU time (in seconds) are given in Table 5.2. Again we notice that the CPU times increase only modestly with problem size.

Number of clusters	BL	DIS	CPU time (s)
2	56	13935.90	7
3	56	3483.86	49
4	56	3483.86	58

Table 5.2: Optimal solution values and CPU times for the 16×16 study region

Figures 5.6(a–c) show the clusters selected for conservation when $C = 2$, $C = 3$, and $C = 4$, respectively. Even though $C = 2$ specifies just one real cluster, model (P_2) identifies two disjoint clusters, as seen in Figure 5.6(a). A slightly different set of clusters is produced for $C = 3$. Namely site $(9, 4)$ of Cluster 2 in Figure 5.6(a) has migrated to position $(15, 11)$ of Cluster 3 in Figure 5.6(b). Increasing $C = 3$ to $C = 4$ does not however change the optimal set of clusters. It is important to notice from Table 5.2 that the boundary length 56 remains unchanged as C is varied, whereas the total within cluster distance decreases from 13935.90 to 3483.86 for $C = 3$ and $C = 4$, respectively. When $C = 2$, we have one real cluster and the distance is measured within and between the two identified components of the single real Cluster 2 (Figure 5.6(a)). But when $C = 3$ and $C = 4$, the distance is measured only within each of the two identified clusters, namely, Cluster 2 and Cluster 3 (Figures 5.6(b–c)). This explains why increasing the number of clusters significantly decreases the optimal within cluster distance DIS.

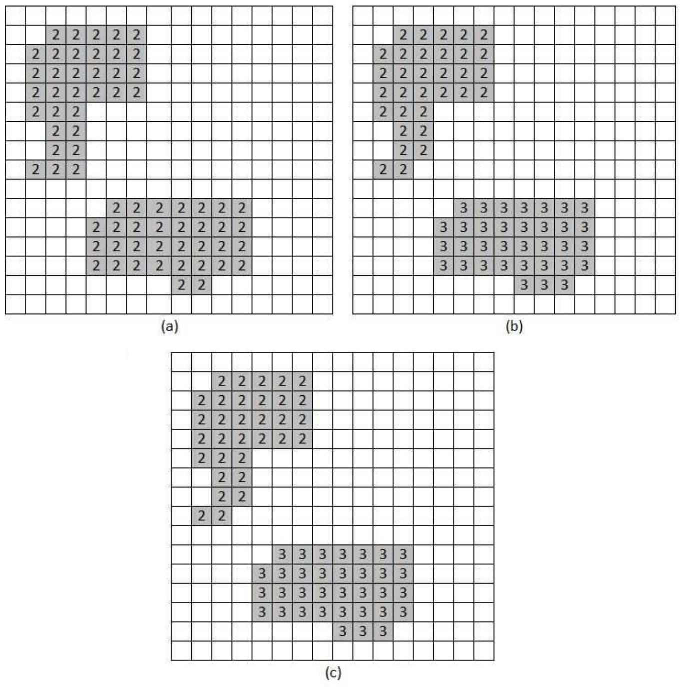


Figure 5.6: Optimal solutions using $C = 2, 3, 4$ for the 16×16 study region

In this example, increasing the number of allowed clusters C does not change the total boundary length BL. Also we note that the optimal solution stays the same for C sufficiently large (here $C \geq 3$). These properties can be shown to hold more generally using model (P_2).

Since model (P_2) produces optimal solutions with the same boundary length for all $C \geq 2$ and since the optimal solution obtained for $C = 2$ provides reasonable clusters, we use $C = 2$ in our models instead of using larger values of C. This will enable us to reduce the required CPU time significantly, yet identifying a near-optimal clustering.

5.3.3 Varying the coverage of species

In this section we discuss how the optimal solution behaves when the required number of species of each type is changed. We illustrate this using the data set presented in [83]. In this example, 16 hypothetical species are distributed across a reserve system containing

100 sites. Since our formulation assumes a border of non-selected sites, we therefore treat this test problem as a 12×12 grid. We now consider two scenarios.

Case 1

First suppose we need to cover at least one of each species. The optimal solution for this case when $C = 2$ is depicted in Figure 5.7. Model (P_2) identifies a solution with three clusters, which is the same solution as that reported in [83]. This same clustering would have been obtained had we used $C > 2$ in our model. Also, it should be mentioned that increasing the maximum number of selected sites UB from 3 to 30 does not change the optimal solution.

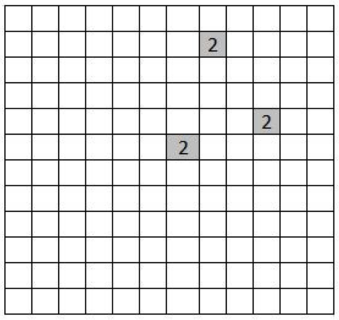


Figure 5.7: Optimal solution using $C = 2$ with each species covered at least once

Case 2

Now suppose we wish to cover at least two of each species. For this case we obtain three different solutions as the maximum number of selected sites is varied for $C = 2$. These optimal solutions are depicted in Figure 5.8, when UB is specified as 6, 7–8, and > 8 respectively. As UB increases, the clusters become fewer in number and the reserve system becomes more compact in shape.

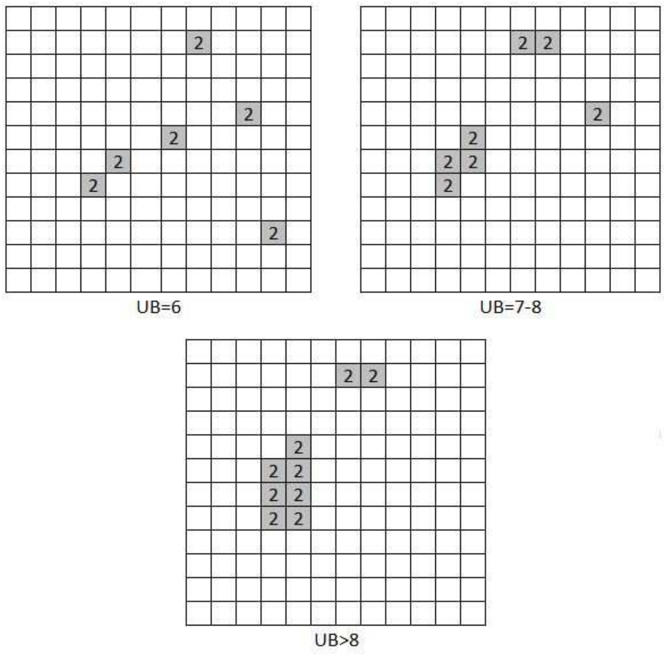


Figure 5.8: Optimal solutions using $C = 2$ with each species covered at least twice

5.4 Oregon data set

In this section we consider the performance of model (P_2) on the Oregon Terrestrial Vertebrate data set reported in [19], in which a grid based distribution map was created using coverage of 635 km^2 hexagons. There are 441 sampling units containing 426 terrestrial vertebrate species. For our study, we selected a study region defined by an 11×16 hexagonal grid. This study region consists of 176 hexagonal units containing 337 species. Since our formulation assumes a border of non-selected sites, we therefore treat this reserve system as a 13×18 hexagonal grid.

We summarize the results of applying optimization model (P_2) to this hexagonal system. We consider two scenarios.

5.4.1 Varying the species to be covered

Here we study how the optimal solution behaves when the species to be covered is changed from rare to common. Specifically, the species to be covered varies from protecting only those species present in exactly one site to protecting all species. Let FB denote the specified *frequency bound*, giving an upper bound on the number of sites in which species can be present. Table 5.3 shows properties of the optimal configuration obtained using $C = 2$ and UB = 60, as FB is varied. We record the optimal boundary length BL, the optimal within cluster distance DIS, and the CPU time taken for each case.

FB	Number of species	BL	DIS	CPU time (s)
1	13	70	614	5
2	18	90	1066	27
3	22	96	1529	48
4	30	104	1805	79
5	38	104	1956	81
10	55	108	2130	119
13	69	108	9817	192
176 (all sites)	337	108	9817	195

Table 5.3: Optimal solution values and CPU times for the 13×18 hexagonal grid when FB is varied

Figures 5.9(a–h) display the optimal clusters found for each case and the newly added sites are shown in black. For example, to cover only the rarest species (FB = 1), the optimal BL is 70 and the optimal DIS is 614. It should be mentioned that as we increase FB from 5 to 9 sites, or from 10 to 12 sites, the optimal solution stays the same. Also, once FB is 13 or greater the same optimal solution persists. In every case, the optimal solution used the maximum number of sites (60). Further, notice that when FB is changed from 4 to 5 sites (Figures 5.9(d–e)) or from 10 to 13 sites (Figures 5.9(f–g)), the solution has the same optimal BL but an increased optimal DIS. This occurs since in covering more species, the optimal solution uses more sites and so the within cluster distance is increased. Yet by adding new sites, we can improve the overall compactness and shape of the solution. Finally, while the solution time grows with the size of the formulated model, it still remains modest (always less than 4 minutes).

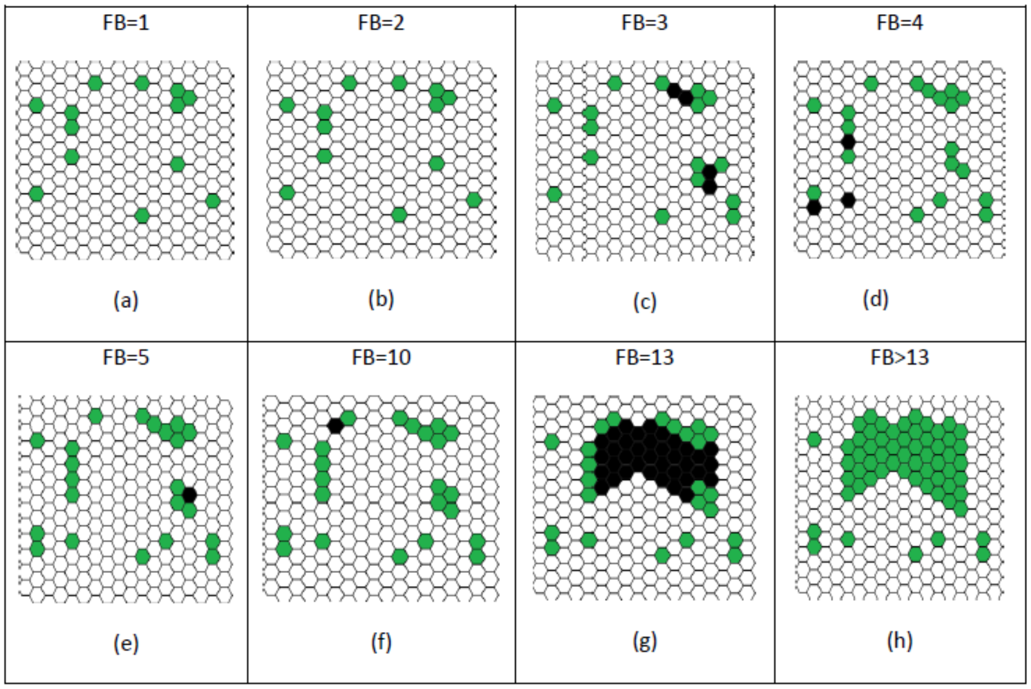


Figure 5.9: Optimal solutions using $C = 2$ for the 13×18 hexagonal grid when FB is varied

5.4.2 Varying UB

Now we discuss how the optimal solution changes when the maximum number of selected sites is changed. We consider the three cases UB = 60, UB = 65, and UB = 70. Table 5.4 shows characteristics of the optimal solutions for these cases, all obtained using $C = 2$ and covering all species.

UB	BL	DIS	CPU time (s)
60	108 (60 sites)	9817	195
65	106 (65 sites)	11307	236
70	104 (68 sites)	12197	181

Table 5.4: Optimal solution values and CPU times for the 13×18 hexagonal grid when UB is varied

Figures 5.10(a–c) show the optimal clusterings obtained for these cases. When UB = 60 the optimal solution uses 60 sites with a optimal BL of 108 and a DIS of 9817. When UB = 65 the optimal solution uses 65 sites; notice that site (10,4) is removed from Figure 5.10(a) and new six sites (shown in black) are added to the new solution in Figure 5.10(b). Even though this solution uses more sites, the optimal BL of 106 improves upon the optimal BL of 108 in the previous case. When UB = 70 the optimal solution only uses 68 sites; the three newly added sites are shown in black in Figure 5.10(c). The optimal BL decreases to 104 and the optimal DIS increases to 12197. This example again illustrates that as UB is increased the optimal solution is improved in terms of BL and the selected region for the optimal solution becomes more compact.

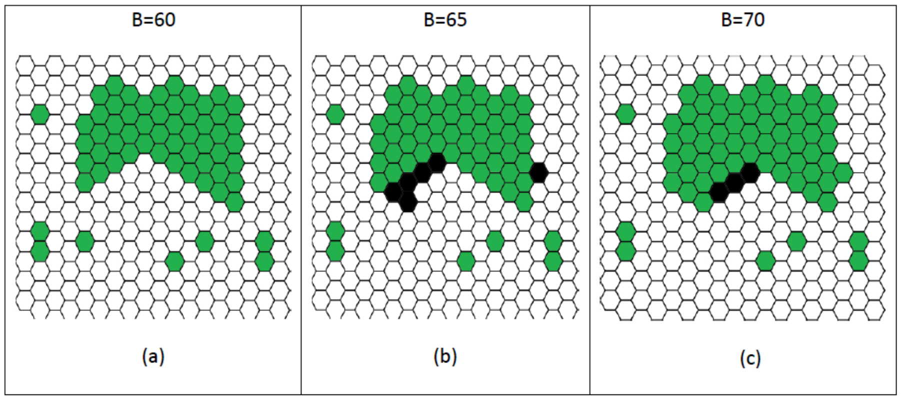


Figure 5.10: Optimal solutions using $C = 2$ for the 13×18 hexagonal grid when UB is varied

5.5 Conclusions and Extensions

This paper studies the design of spatially compact reserve systems, which addresses limitations of the standard SCP and MCP covering approaches. We develop a hierarchical optimization model that organizes reserve sites into a relatively small number of compact groups (clusters). This model explicitly considers two factors: minimizing the boundary length of all selected clusters and minimizing the total distance between all pairs of sites within each cluster. Each of these objectives is important for the long-term success of the planned reserve system. We argue that when creating desirable clusters the boundary length is more important than the total within cluster distance. This naturally leads to a hierarchical optimization model that gives priority to minimizing the boundary length, and then minimizing the total within cluster distance as a secondary objective. These hierarchical objectives are combined into a single objective function using an appropriate weight U.

Our initial formulation (P_1) is expressed as a integer programming problem with a nonlinear, nonconvex objective function. To solve the model more efficiently we converted the initial formulation into a linear mixed integer programming problem (P_2), having the same optimal solution as model (P_1). The computationally more efficient model (P_2) was then used in all our subsequent numerical investigations. When applied to various data sets this model provided reasonable clusters. We also studied how the optimal solutions behave when the allowable number of clusters C, the maximum number of selected sites UB, and the specified frequency bound FB are changed. Further, we showed that $C = 2$ provides a clustering with the optimal boundary length although it may not provide the optimal within cluster distance. Since our first priority is minimizing the boundary length of the selected clusters, we can use $C = 2$ to obtain a meaningful set of clusters, while reducing the required CPU time significantly.

For large reserve design problems it may be difficult to find an optimal solution of model (P_2) in a reasonable amount of time even for $C = 2$. Consequently, the use of heuristic algorithms (e.g., see [16, 55]) should be explored. We briefly outline a heuristic procedure based on a linear relaxation of model (P_2). Namely, the binary constraints $X_{cij} \in \{0, 1\}$ are replaced by $0 \leq X_{cij} \leq 1$. We then identify those sites (i, j) having $X_{cij} = 1$ and $c > 1$ in the relaxed solution. This partial solution satisfies the budget constraint (10) but in general does not satisfy the species coverage constraints (9). To achieve feasibility in model (P_2), we consider all non-selected sites adjacent to existing selected sites as possible candidates for inclusion. These candidates are prioritized based on the following factors: the additional species coverage provided, the change in boundary length, the fractional value of X_{cij}, and the change in within cluster distance.

Once a feasible solution is obtained, it can be improved by applying an interchange heuristic. Namely, we can swap a currently selected site with another non-selected site that is adjacent to an existing cluster, after ensuring that this modification maintains feasibility. We can therefore improve the current solution by considering our two fundamental criteria. First we consider the reduction in boundary length due to such an interchange. If several interchanges yield the same reduction in boundary length then we consider the reduction in total within cluster distance achieved by the interchange. Preliminary experience with such a heuristic has been encouraging and research is ongoing to explore this approach.

Appendices

Appendix A MATLAB codes for Chapter 3

Appendix A includes the MATLAB codes for implementing Algorithm 1 and Algorithm 2 described in Chapter 3. These include the generic algorithm and two procedures get-weighted-sum-sol() and get-max-sum-sol().

A.1 Generic algorithm for the MOSCP

The generic algorithm uses two procedures. The generic algorithm and get-weighted-sum-sol() correspond to Algorithm 1 and the generic algorithm and get-max-sum-sol() correspond to Algorithm 2. The characteristics of the test problem are the input parameters for the main algorithm.

```
clear;clc;
addpath C:\Lakmali\weightedsetcovering
data_weighted=load('C:\Lakmali\weightedsetcovering\2scp11A\2scp11A.txt');
[num_item num_set]=size(data_weighted);
data=data_weighted(1:num_item-2,1:num_set);% pick the data part
C=data_weighted(num_item-1:num_item,1:num_set);% pick the objective function
num_item=num_item-2; % reset the number of items
%true_1=[];true_2=[]; wei_1=[];wei_2=[];
lambda_1=1;
generated_num_items=to_get_true_num_items(data,num_item);
dummy_C=C;
dummy_data=data;
solution_set=[];
solution_set_obj=[];
if (generated_num_items==num_item)
    while (lambda_1>=0)
        [selected_set,cost_1,cost_2,z_w]=get_weighted_sum_sol(data,C,num_item,dummy_C,lambda_1,(1-lambda_1));
        %[selected_set,cost_1,cost_2,z_w]=get_max_sum_sol(data,C,num_item,dummy_C,lambda_1,(1-lambda_1));
        solution_set=[solution_set ;lambda_1, cost_1,cost_2];
        lambda_1=lambda_1-0.1;
        C=dummy_C;
        data=dummy_data;
    end
else
  disp(sprintf('not a good data set'))
end
get_unique_solution(solution_set)
solution_set
plot(solution_set(:,2),solution_set(:,3),'ro')
```

A.2 Implementation of Algorithm 1

The get-weighted-sum-sol() procedure provides feasible solutions to the MOSCP based on the scalar cost described in Section 3.3. Characteristics of the test problem and the direction λ are the main input parameters for this procedure. For each direction λ, the main method generates feasible solutions using get-weighted-sum-sol() procedure.

```
function [selected_set,cost_1,cost_2,z_w]=get_weighted_sum_sol(data,C,num_item,dummy_C,lambda_1,lambda_2)
selected_set=[]; covered_items=zeros(1,num_item);
C_1=C(1,:);C_2=C(2,:);
%lambda_2=1-lambda_1;
C=lambda_1*C_1+ lambda_2*C_2;
cost_1=0; cost_2=0;
[cost_ratio,num_coverd]=construct_ratio_array(data,C);% gives the ratio for each set wrt each obj
[weighted_preference_value,weighted_preference_value_position]=get_weighted_preference_value(cost_ratio);
currently_covered_items=data(:,weighted_preference_value_position);
covered_items=to_get_currently_covered_items(currently_covered_items,covered_items);
data=to_update_data_set(data,weighted_preference_value_position);
selected_set=[selected_set weighted_preference_value_position];
cost_1=cost_1+ dummy_C(1,weighted_preference_value_position);
cost_2=cost_2+  dummy_C(2,weighted_preference_value_position);

while (sum(covered_items)~=num_item)
    [cost_ratio,num_coverd]=construct_ratio_array(data,C);% gives the ratio for each set wrt each obj
    [weighted_preference_value,weighted_preference_value_position]=get_weighted_preference_value(cost_ratio);
    currently_covered_items=data(:,weighted_preference_value_position);
    currently_covered_items=to_get_currently_covered_items(currently_covered_items,covered_items);
    covered_items=to_get_currently_covered_items(currently_covered_items,covered_items);
    data=to_update_data_set(data,weighted_preference_value_position);
    selected_set=[selected_set weighted_preference_value_position];
    cost_1=cost_1+ dummy_C(1,weighted_preference_value_position);
    cost_2=cost_2+  dummy_C(2,weighted_preference_value_position);
end
selected_set;
z_w=lambda_1*cost_1+lambda_2*cost_2;
```

A.3 Implementation of Algorithm 2

The get-max-sum-sol() procedure provides feasible solutions to the MOSCP based on the vector cost described in Section 3.3. Characteristics of the test problem and the direction λ are the main input parameters for this procedure. For each direction λ, the main method generates feasible solutions using get-max-sum-sol() procedure.

```
function [selected_set,cost_1,cost_2,z_w]=get_max_sum_sol(data,C,num_item,dummy_C,lambda_1,lambda_2)
selected_set=[]; covered_items=zeros(1,num_item);
C_1=C(1,:);C_2=C(2,:);
cost_1=0; cost_2=0;
[cost_ratio,num_coverd]=construct_ratio_array(data,C);% gives the ratio for each set wrt each obj
[max_preference_value,max_preference_value_position]=get_max_preference_value(cost_ratio);
currently_covered_items=data(:,weighted_max_value_position);
covered_items=to_get_currently_covered_items(currently_covered_items,covered_items);
data=to_update_data_set(data,max_preference_value_position);
selected_set=[selected_set max_preference_value_position];
cost_1=cost_1+ dummy_C(1,max_preference_value_position);
cost_2=cost_2+ dummy_C(2,max_preference_value_position);

while (sum(covered_items)~=num_item)
    [cost_ratio,num_coverd]=construct_ratio_array(data,C);% gives the ratio for each set wrt each obj
    [max_preference_value,max_preference_value_position]=get_max_preference_value(cost_ratio);
    currently_covered_items=data(:,max_preference_value_position);
    currently_covered_items=to_get_currently_covered_items(currently_covered_items,covered_items);
    covered_items=to_get_currently_covered_items(currently_covered_items,covered_items);
    data=to_update_data_set(data,max_preference_value_position);
    selected_set=[selected_set max_preference_value_position];
    cost_1=cost_1+ dummy_C(1,max_preference_value_position);
    cost_2=cost_2+  dummy_C(2,max_preference_value_position);
end
selected_set;
z_w=[lambda_1*cost_1 lambda_2*cost_2];
```

Appendix B MATLAB codes for Chapter 4

Appendix B includes the MATLAB codes for implementing the Add-Improve Algorithm described in Chapter 4. These include the main method and the four procedures, get-min-data(), get-weighted-lb-sol(), get-cheby-lb-sol() and drop-redundant().

B.1 Implementation of the Add-Improve Algorithm

The main method uses four procedures and it corresponds to Algorithm 1 of Chapter 4. The set of different λ vectors defined by the decision maker and the characteristics of the test problem are the input parameters for the main method. At the beginning of the algorithm, the main method calls get-min-data() procedure to obtain the D matrix described in Section 4.4. Then, for each λ, it generates feasible solutions using get-weighted-lb-sol() and get-cheby-lb-sol() procedures and improves the feasible solutions using drop-redundant() procedure.

```
clear;clc;
addpath C:\Lakmali\weightedsetcovering
data_weighted=load('C:\Lakmali\weightedsetcovering\2scp41A\2scp41A.txt');
[num_item num_set]=size(data_weighted); data=data_weighted(1:num_item-2,1:num_set);% pick the data part
C=data_weighted(num_item-1:num_item,1:num_set);% pick the objective function
num_item=num_item-2; % reset the number of items
lambda_1=1;
solution_set=[]; new_greedy=[]; w_new_greedy=[]; c_new_greedy=[];
dummy_C=C; w_solution_set=[];c_solution_set=[];obj=[]; sol=[];
min_data=get_min_data(C,data);
rhs= -ones(num_item,1) ;
[x_1, fval_1] = linprog(C(1,:), -data, rhs,[],[], zeros(num_set,1), ones(num_set,1),[]); % solve LP model
[x_2, fval_2] = linprog(C(2,:), -data, rhs,[],[], zeros(num_set,1), ones(num_set,1),[]); % solve LP model
tic;
ideal=[fval_1,fval_2];
while (lambda_1>=0)
    lambda_2=1-lambda_1;
    X=zeros(1,num_set);cost_1=0;cost_2=0; covered_items=zeros(1,num_item);
    [sol,cost_1,cost_2]=get_weighted_lb_sol(data,C,lambda_1,lambda_2,min_data,X,cost_1,cost_2,covered_items);
    w_solution_set=[w_solution_set ;cost_1,cost_2];
    w_new_greedy=[w_new_greedy; sol lambda_1 ];
    lambda_1=lambda_1-0.1;
    C=dummy_C;
end
```

```
[num col]=size(w_new_greedy)
for i=1:num
    feasible_solution=w_new_greedy(i,1:end-1);
    improved_solution=drop_redundant(data,feasible_solution,C,w_new_greedy(i,end:end));
    [val1,val2]=cplex_z1_z2(improved_solution, C,num_set);
    obj=[obj; val1 val2];
    sol=[sol; improved_solution];
end
lambda_1=1;
while (lambda_1>=0)
    [sol,cost_1,cost_2]=get_cheby_lb_sol(data,C,lambda_1,(1-lambda_1),min_data);
    c_solution_set=[c_solution_set ;cost_1,cost_2];
    c_new_greedy=[c_new_greedy;sol lambda_1];
    lambda_1=lambda_1-0.1;
    C=dummy_C;
end
[num col]=size(c_new_greedy)
for i=1:num
    feasible_solution=c_new_greedy(i,1:end-1);
    size(feasible_solution)
    improved_solution=drop_redundants(data,feasible_solution,C,c_new_greedy(i,end:end));
    [val1,val2]=cplex_z1_z2(improved_solution, C,num_set);
    obj=[obj; val1 val2];
    sol=[sol; improved_solution];
end
[num col]=size(relaxed);
for i=1:num
    feasible_solution=relaxed(i,:);
    improved_solution=drop_redundants(data,feasible_solution,C);
    [val1,val2]=cplex_z1_z2(improved_solution, C,num_set);
    obj=[obj; val1 val2];
    sol=[sol; improved_solution];
end
toc;
dlmwrite('C:\Lakmali\weightedsetcovering\2scp41A\improved_ob_value.txt',obj,'delimiter','\t');
dlmwrite('C:\Lakmali\weightedsetcovering\2scp41A\improved_solution.txt',improved_solution,'delimiter','\t');
```

The procedure get-min-data() determines the best set to cover each item with respect to each objective function. This procedure uses the characteristic of the test problem as the input parameters.

```
function min_data=get_min_data(C,data);
min_data=[]; [num_item, num_set]=size(data)
for i=1:num_item
    min1=100000; min2=100000;
    for j=1:num_set
```

```
        if(data(i,j)==1)
            if (C(1,j)/sum(data(:,j))<min1)
                min1=C(1,j)/sum(data(:,j));
                j1=j;
            end
            if (C(2,j)/sum(data(:,j))<min2)
                min2=C(2,j)/sum(data(:,j));
                j2=j;
            end
         end
    end
    min_data=[min_data; C(1,j1) C(2,j2) j1 j2];
end
```

The procedure get-weighted-lb-sol() constructs the feasible solutions of the MOSCP by estimating the cost of each set as described in equation (4.7) in Section 4.3.

```
function [sol,cost_1,cost_2]=get_weighted_lb_sol(data,C,lambda_1,lambda_2,...
                             ...min_data, X,cost_1,cost_2,covered_items)
[num_item,num_set]=size(data); selected_set=[];
%covered_items=zeros(1,num_item);
dummy_C=C; Dummy_D=min_data;
%X=zeros(1,num_set);
Cost=zeros(1,num_set);
%cost_1=0; cost_2=0;
while (sum(covered_items)~=num_item)
    for j=1:num_set
        min1=60000;min2=600000;
        if(X(j)==0)
            dummyset1=zeros(1,num_set);
            dummyset2=zeros(1,num_set);
            for i=1:num_item
                if(covered_items(i)==0 && data(i,j)==0)
                    if(dummyset1(min_data(i,3))~=inf)
                        C(1,j)=C(1,j)+(min_data(i,1));
                        dummyset1(min_data(i,3))=inf;
                    end
                    if(dummyset2(min_data(i,4))~=inf)
                        C(2,j)=C(2,j)+(min_data(i,2));
                        dummyset2(min_data(i,4))=inf;
                    end
                end
            end
            Cost(j)= (lambda_1*(C(1,j))+lambda_2*(C(2,j)));
        else
            Cost(j)= inf;
        end
    end
    [min_cost, position_min_cost]=min(Cost);

    C=dummy_C;
    %selected_set=[selected_set position_min_cost];
    currently_covered_items=data(:,position_min_cost);
    covered_items=to_get_currently_covered_items(currently_covered_items,covered_items);
    X(position_min_cost)=1;
    cost_1=cost_1+ C(1,position_min_cost);
    cost_2=cost_2+ C(2,position_min_cost);
end
sol=X;
```

The procedure get-cheby-lb-sol() constructs the feasible solutions of the MOSCP by estimating the cost of each set as described in equation (4.8) in Section 4.3.

```
function [sol,cost_1,cost_2]=get_cheby_lb_sol(data,C,lambda_1,lambda_2,min_data)
[num_item,num_set]=size(data); selected_set=[]; covered_items=zeros(1,num_item);
dummy_C=C; X=zeros(1,num_set);Cost=zeros(1,num_set);
cost_1=0; cost_2=0;
while (sum(covered_items)~=num_item)
    for j=1:num_set
        min1=60000;min2=600000;
        if(X(j)==0)
            dummyset1=zeros(1,num_set);
            dummyset2=zeros(1,num_set);
             for i=1:num_item
                if(covered_items(i)==0 && data(i,j)==0)
                    if(dummyset1(min_data(i,3))~=inf)
                    C(1,j)=C(1,j)+(min_data(i,1));
                    dummyset1(min_data(i,3))=inf;
                    end
                    if(dummyset2(min_data(i,4))~=inf)
                    C(2,j)=C(2,j)+(min_data(i,2));
                    dummyset2(min_data(i,4))=inf;
                    end
             end
           % possible_covered=get_possible_covered(data,covered_items,j);
           Cost(j)=max(lambda_1*(C(1,j)),lambda_2*(C(2,j)));
           %Cost(j)=max(lambda_1*(C(1,j)),lambda_2*(C(2,j)));
        else
           Cost(j)= inf;
        end

     end
    [min_cost, position_min_cost]=min(Cost);
    C=dummy_C;
    %selected_set=[selected_set position_min_cost];
    currently_covered_items=data(:,position_min_cost);
    currently_covered_items=to_get_currently_covered_items(currently_covered_items,covered_items);
    covered_items=to_get_currently_covered_items(currently_covered_items,covered_items);
    X(position_min_cost)=1;
    cost_1=cost_1+ C(1,position_min_cost);
    cost_2=cost_2+ C(2,position_min_cost);
end
sol=X;
```

The drop-redundant() procedure improves feasible solutions provided by get-weighted-lb-sol() and get-cheby-lb-sol() procedures using the concepts discussed in Section 4.4.2. The input parameters are the problem characteristics, feasible solution and its corresponding direction λ.

```
function improved_solution=drop_redundant(data,feasible_solution,C,lambda)
improved_solution=feasible_solution; [num_item,num_set]=size(data); potential_drops=[0];
final_covers=data*feasible_solution';
while(min(final_covers)>=1 && isempty(potential_drops)~=1 )
    potential_drops=[];
    for i=1:num_set
        vec= final_covers-data(:,i);
        if (improved_solution(i)==1 && min(vec)>=1)
            cost= (lambda*C(1,i)+(1-lambda)*C(2,i));
            potential_drops=[potential_drops; cost i];
            %final_covers=final_covers-data(:,i);
            %improved_solution(i)=0;
        end
    end

   if (min(final_covers)>=1 && isempty(potential_drops)~=1)
    potential_drops=sortrows(potential_drops,-1);
         final_covers=final_covers-data(:,potential_drops(1,2));
        improved_solution(potential_drops(1,2))=0;
   end
end
improved_solution=[improved_solution];
```

Appendix C OPL code for Chapter 5

Appendix C includes the OPL code for implementing the mathematical model (P_2) described in Chapter 5. The implementation is specific to a uniformly partitioned hexagonal region.

C.1 OPL code for Hexagonal Linear Model

```
/*********************************************
 * OPL 12.4 Hexagonal Linear Model
 * Author: Lakmali Weerasena
 * Creation Date: Jan 5, 2013 at 9:25:01 AM
 *********************************************/

int n=13; // Number of rows in the Grid
int m=23; // Number of columns in the Grid
int UP=10000000; //Upper Bound on Distance
int s=60; // maximum number of sites
int c=2; // number of clusters
int b=343; // number of species
int a=4459; // number of rows of the imported data set
range row=1..n;
range column=1..m;
range clusters=1..c;
range num_spec=1..b;
range block_row=1..a;
int areaofs[block_row,column]=...; // imported data set
int required_spec[num_spec]=...; // required coverage
int x_temp[1..c*n][1..m];
//dvar int x[row][column][clusters];
string range1;
float obj_final;
dvar boolean X[clusters][row][column]; //  DECISION VARIABLES X;
dvar float Y[clusters][row][column][row][column];//  DECISION VARIABLES Y;
// All Constraints
constraint ct1;  constraint ct2;  constraint ct3;  constraint ct4;  constraint ct5;  constraint ct6;
constraint ct7;  constraint ct8;  constraint ct9;  constraint ct10; constraint ct11; constraint ct12;
constraint ct13; constraint ct14; constraint ct15; constraint ct16; constraint ct17; constraint ct18;
constraint ct19; constraint ct20; constraint ct21; constraint ct22; constraint ct23; constraint ct24;
constraint ct25; constraint ct26; constraint ct27; constraint ct28; constraint ct29; constraint ct30;
constraint ct31; constraint ct32; constraint ct33; constraint ct34; constraint ct35; constraint ct36;
constraint ct37; constraint ct38; constraint ct39; constraint ct40; constraint ct41; constraint ct42;
constraint ct43; constraint ct44; constraint ct45; constraint ct46; constraint ct47; constraint ct48;
constraint ct49; constraint ct50; constraint ct51; constraint ct52; constraint ct53; constraint ct54;
constraint ct55;
```

```
//  minimize

   dexpr float j_odd =
   // j<=l, i<=k
              sum (c in clusters, i in row,k in row ,j in column,l in column:
                   (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  ((i<k  && j<l)  ) && (j mod 2!=0)  )
                    (maxl(0,(k-i-floor((l-j)/2))) +(l-j))*Y[c,i,j,k,l]

             + sum (c in clusters, i in row,k in row ,j in column,l in column:
                   (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  ((i==k  && j<l) || (i<k  && j==l) ) && (j mod 2!=0)  )
                    (maxl(0,(k-i-floor((l-j)/2))) +(l-j))*Y[c,i,j,k,l]
   // j<=l, i> k
             + sum (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i>k  &&  j<=l  && (j mod 2!=0))
                    (maxl(0,(i-k-ceil((l-j)/2)))+(l-j))* Y[c,i,j,k,l];

   dexpr float j_even =
    // j<=l, i<=k
             sum (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i<k  && j<l && (j mod 2 ==0)  )
                    (maxl(0,(k-i-ceil((l-j)/2)))+(l-j)) * Y[c,i,j,k,l]

            +sum (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  ((i==k  && j<l) || (i<k  && j==l) ) && (j mod 2 ==0)  )
                    (maxl(0,(k-i-ceil((l-j)/2)))+(l-j)) * Y[c,i,j,k,l]
   // j<=l, i>k
             + sum (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i>k  &&  j<=l  && (j mod 2==0))
                    (maxl(0,(i-k-floor((l-j)/2)))+(l-j)) * Y[c,i,j,k,l];

  dexpr float l_odd =
  // j>l, i<=k
              sum (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i<=k  && j>l  && (l mod 2 !=0))
                    (maxl(0,(k-i-ceil((j-l)/2)))+(j-l)) * Y[c,i,j,k,l]
   // j>l, i > k
             + sum (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
```

```
                  &&  i>k  &&  j>l   && (l mod 2!=0))
                  (maxl(0,(i-k-floor((j-l)/2)))+(j-l)) * Y[c,i,j,k,l];

   // j>l, i<=k
   dexpr float l_even =
             sum (c in clusters, i in row,k in row ,j in column,l in column:
                (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                  && i<=k && j>l && (l mod 2==0))
                  (maxl(0,(k-i-floor((j-l)/2)))+(j-l)) * Y[c,i,j,k,l]
   // j>l, i>k
            + sum (c in clusters, i in row,k in row ,j in column,l in column:
                  (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                  &&  i>k  &&  j>l  && (l mod 2==0) )
                  (maxl(0,(i-k-ceil((j-l)/2)))+(j-l)) * Y[c,i,j,k,l];

minimize  (j_odd+j_even+l_odd+l_even) // objective function

+ UP* sum ( c in clusters, i in row ,j in column :
    c==1 && i>2 && i<n-1 && j>2 && j<m-1  &&(j mod 2==0) )
   (6*X[1,i,j]-Y[1,i,j,i-1,j]-Y[1,i,j,i,j+1]-Y[1,i,j,i+1,j+1]-Y[1,i,j,i+1,j]-Y[1,i,j,i+1,j-1]-Y[1,i,j,i,j-1])

+ UP* sum ( c in clusters, i in row, j in column:
     c==1 && i>2 && i<n-1 && j>2 && j<m-1 && (j mod 2!=0) )
   (6*X[1,i,j]-Y[1,i,j,i-1,j]-Y[1,i,j,i-1,j+1]-Y[1,i,j,i,j+1]-Y[1,i,j,i+1,j]-Y[1,i,j,i,j-1]-Y[1,i,j,i-1,j-1])


//[1] neighbours of  (real) left upper  corner X(2,2)
            + UP* sum ( c in clusters, i in row ,j in column :
                 c==1 && i==2 && j==2 )
                 (3*X[1,i,j]-Y[1,i,j,i,j+1]-Y[1,i,j,i+1,j+1]-Y[1,i,j,i+1,j])

//[2] neighbour of X(2,1)
            + UP* sum ( c in clusters, i in row ,j in column :
                 c==1 && i==2 && j==1 )
                 (1*X[1,i,j]-Y[1,i,j,i,j+1])

//[3] neighbours of X(1,2)
            +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==1 && j==2   )
                (2*X[1,i,j]-Y[1,i,j,i+1,j]-Y[1,i,j,i+1,j+1])

//[4] neighbours of X(2,m-1)  (real) right upper corner (m is even)
            +UP*sum ( c in clusters, i in row, j in column :
                c==1 && i==2 && j==m-1  && (m mod 2==0) )
                (2*X[1,i,j]-Y[1,i,j,i+1,j]-Y[1,i,j,i,j-1])

//[5] neighbour of X(1,m) (m is  even)
```

```
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==1 && j==m && (m mod 2==0) )
                 (1*X[1,i,j]-Y[1,i,j,i+1,j-1])

//[6] neighbours of X(1,m-1) (m is even)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==1 && j==m-1 && (m mod 2==0) )
                 (1*X[1,i,j]-Y[1,i,j,i+1,j])

//[7] neighbours of right upper corner X(2,m-1)  (m is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                  c==1 && i==2 && j==m-1 && (m mod 2!=0)  )
                 (3*X[1,i,j]-Y[1,i,j,i+1,j]-Y[1,i,j,i+1,j-1]-Y[1,i,j,i,j-1])

//[8] neighbours of X(1,m-1) (m is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==1 && j==m-1 && (m mod 2!=0) )
                 (2*X[1,i,j]-Y[1,i,j,i+1,j-1]-Y[1,i,j,i+1,j])

//[9] neighbour of X(2,m)(m is  odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==2 && j==m  && (m mod 2!=0) )
                 (1*X[1,i,j]-Y[1,i,j,i,j-1])



//[10] neighbours of middle top X(2,j) (j is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==2 && j>2 && j<m-1 && (j mod 2!=0))
                 (3*X[1,i,j]-Y[1,i,j,i,j+1]-Y[1,i,j,i+1,j]-Y[1,i,j,i,j-1])

//[11] neighbour of midlle top boundary X(1,j) ( j is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==1 && j>2 && j<m-1 && (j mod 2!=0))
                 (1*X[1,i,j]-Y[1,i,j,i+1,j])

//[12] neighbours of midlle top X(2,j) (j is even)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                 (5*X[1,i,j]-Y[1,i,j,i,j+1]-Y[1,i,j,i+1,j+1]-Y[1,i,j,i+1,j]-Y[1,i,j,i+1,j-1]-Y[1,i,j,i,j-1])

//[13]  neighbours of midlle top boundary X(1,j) (j is even)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==1 && j>2 && j<m-1 && (j mod 2==0))
                 (3*X[1,i,j]-Y[1,i,j,i+1,j-1]-Y[1,i,j,i+1,j]-Y[1,i,j,i+1,j+1])

//[14] neighbours of left  X(i,2)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1  && j==2)
                 (4*X[1,i,j]-Y[1,i,j,i-1,j]-Y[1,i,j,i,j+1]-Y[1,i,j,i+1,j+1]-Y[1,i,j,i+1,j])


//[15] neighbours left boundary X(i,1)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<=n-1  && j==1)
                 (2*X[1,i,j]-Y[1,i,j,i-1,j+1]-Y[1,i,j,i,j+1])

//[16] neighbours of right  X(i,m-1) (m is even)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                 (4*X[1,i,j]-Y[1,i,j,i-1,j]-Y[1,i,j,i-1,j-1]-Y[1,i,j,i,j-1]-Y[1,i,j,i+1,j])

//[17] neighbours of  right boundary  X(i,m) ( m is even)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i>=2 && i<n-1 &&  j==m && (m mod 2==0))
                 (2*X[1,i,j]-Y[1,i,j,i,j-1]-Y[1,i,j,i+1,j-1])

//[18] neighbours of right X(i,m-1) (m is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 (4*X[1,i,j]-Y[1,i,j,i-1,j]-Y[1,i,j,i,j-1]-Y[1,i,j,i+1,j-1]-Y[1,i,j,i+1,j])

//[19] neighbours of  right boundary X(i,m) (m is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<=n-1 &&  j==m && (m mod 2!=0))
                 (2*X[1,i,j]-Y[1,i,j,i-1,j-1]-Y[1,i,j,i,j-1])

//[20] neighbours of  bottom left corner X(n-1,2)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==2 )
                 (2*X[1,i,j]-Y[1,i,j,i-1,j]-Y[1,i,j,i,j+1])

//[21] neighbour of  bottom left boundary  X(n,1)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==n && j==1 )
                  (1*X[1,i,j]-Y[1,i,j,i-1,j+1])

//[22] neighbour of bottom left boundary  X(n,2)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==n && j==2 )
                 (1*X[1,i,j]-Y[1,i,j,i-1,j])

//[23] neighbours of bottom right corner X(n-1,m-1) (m is even)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2==0) )
```

```
                  (3*X[1,i,j]-Y[1,i,j,i-1,j]-Y[1,i,j,i-1,j-1]-Y[1,i,j,i,j-1])

//[24] neighbour of bottom right boundary X(n-1,m) (m is even)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m && (m mod 2==0) )
                  (1*X[1,i,j]-Y[1,i,j,i,j-1])

//[25] neighbour of bottom right boundary  X(n,m-1) (m is even)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==n && j==m-1 && (m mod 2==0) )
                  (2*X[1,i,j]-Y[1,i,j,i-1,j]-Y[1,i,j,i-1,j-1])

//[26] neighbour of bottom right corner X(n-1,m-1) (m is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2!=0) )
                  (2*X[1,i,j]-Y[1,i,j,i-1,j]-Y[1,i,j,i,j-1])

//[27] neighbour of bottom right boundary X(n,m)(m is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==n && j==m && (m mod 2!=0) )
                  (1*X[1,i,j]-Y[1,i,j,i-1,j-1])

//[28] neighbour of bottom right boundary X(n,m-1)(m is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 && i==n && j==m-1 && (m mod 2!=0) )
                  (1*X[1,i,j]-Y[1,i,j,i-1,j])

//[29] neighbours of bottom X(n-1,j)(j is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 (5*X[1,i,j]-Y[1,i,j,i,j-1]-Y[1,i,j,i-1,j-1]-Y[1,i,j,i-1,j]-Y[1,i,j,i-1,j+1]-Y[1,i,j,i,j+1])

//[30]neighbours of bottom boundary X(n,j) ( j is odd)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 &&  i==n && j> 2 && j< (m-1)  && (j mod 2!=0))
                 (3*X[1,i,j]-Y[1,i,j,i-1,j-1]-Y[1,i,j,i-1,j]-Y[1,i,j,i-1,j+1])

//[31] neighbours of bottom X(n-1,j) (j is even)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2==0))
                 (3*X[1,i,j]-Y[1,i,j,i,j-1]-Y[1,i,j,i-1,j]-Y[1,i,j,i,j+1])

//[32] neighbour of bottom boundary X(n,j)(j is even)
             +UP*sum ( c in clusters, i in row, j in column :
                 c==1 &&  i==n && j> 2 && j< (m-1)  && (j mod 2==0))
                 (1*X[1,i,j]-Y[1,i,j,i-1,j])  ;
 subject to {
```

```
//maximum number of cells to be covered;

ct1=   sum  (c in clusters,i in row,j in column :
              c!=1 && i!=1 && i!=n && j!=1&& j!=m)
              X[c,i,j] <=s;
//-----------------------------------------------------------------------------------;

//EVERY CELL IN ONE CLUSTER;

ct2=   forall (i in row, j in column)
             sum ( c in clusters)
               X[c,i,j] ==1;



// coverage for each species
 ct3=   forall (b in num_spec)
            sum ( c in clusters, i in row, j in column:
             c!=1 )
              areaofs[i+n*(b-1),j]*X[c,i,j] >=required_spec[b] ;

//cells on the boundary are placed in cluster1;

ct6=   sum (c in clusters, i in row ,j in column: i==1 && c==1) X[c,i,j]==m;
ct7=   sum (c in clusters, i in row ,j in column: i==n && c==1) X[c,i,j]==m;
ct8=   sum (c in clusters, i in row ,j in column: j==1 && c==1) X[c,i,j]==n;
ct9=   sum (c in clusters, i in row ,j in column: j==m && c==1) X[c,i,j]==n;

//SUB CLUSTER ELIMINATION;

//j even;
 ct10=   forall (c1,c2 in clusters, i in row,j in column:
        (i>2) && (i <= n-1) && (j>=2) && (j<=m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2==0))
               X[c1,i,j]+X[c2,i-1,j]<=1;
 ct11=   forall (c1,c2 in clusters, i in row,j in column:
        (i>=2) && (i <= n-1) && (j>=2) && (j<m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2==0))
               X[c1,i,j]+X[c2,i,j+1]<=1;
 ct12=   forall (c1,c2 in clusters, i in row,j in column:
        (i>=2) && (i < n-1) && (j>=2) && (j< m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2==0))
               X[c1,i,j]+X[c2,i+1,j+1]<=1;
 ct13=   forall (c1,c2 in clusters, i in row,j in column:
        (i>=2) && (i < n-1) && (j>=2) && (j<=m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2==0))
               X[c1,i,j]+X[c2,i+1,j]<=1;
 ct14=   forall (c1,c2 in clusters, i in row,j in column:
        (i>=2) && (i < n-1) && (j>=2) && (j<=m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2==0))
               X[c1,i,j]+X[c2,i+1,j-1]<=1;
 ct15=   forall (c1,c2 in clusters, i in row,j in column:
        (i>=2) && (i <= n-1) && (j>=2) && (j<=m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2==0))
               X[c1,i,j]+X[c2,i,j-1]<=1;
//  j odd;
 ct16=   forall (c1,c2 in clusters, i in row,j in column:
        (i>=2) && (i <= n-1) && (j>=2) && (j<=m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2!=0))
               X[c1,i,j]+X[c2,i-1,j]<=1;
 ct17=   forall (c1,c2 in clusters, i in row,j in column:
        (i>2) && (i <= n-1) && (j>=2) && (j<m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2!=0))
               X[c1,i,j]+X[c2,i-1,j+1]<=1;
 ct18=   forall (c1,c2 in clusters, i in row,j in column:
        (i>=2) && (i <= n-1) && (j>=2) && (j<m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2!=0))
               X[c1,i,j]+X[c2,i,j+1]<=1;
 ct19=   forall (c1,c2 in clusters, i in row,j in column:
        (i>=2) && (i < n-1) && (j>=2) && (j<=m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2!=0))
               X[c1,i,j]+X[c2,i+1,j]<=1;
 ct20=   forall (c1,c2 in clusters, i in row,j in column:
        (i>=2) && (i <= n-1) && (j>=2) && (j<=m-1)
               && ( c1<c2) && ( c1!=1) && (c2!=1) && (j mod 2!=0))
               X[c1,i,j]+X[c2,i,j-1]<=1;
 ct21=   forall (c1,c2 in clusters, i in row,j in column:
        (i>=2) && (i <= n-1) && (j>=2) && (j<=m-1)
               && ( c1< c2) && ( c1!=1) && (c2!=1) && (j mod 2!=0))
               X[c1,i,j]+X[c2,i-1,j-1]<=1;

 // From the boundary length part
 // j is even

 ct22= forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1 ) && (j mod 2==0)  )
                    Y[c,i,j,i-1,j]<=X[c,i,j];


 ct28= forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1 ) && (j mod 2==0)  )
                    Y[c,i,j,i-1,j]<=X[c,i-1,j];
```

```
forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                         j>2 && j<m-1 ) && (j mod 2==0)  )
                   -Y[c,i,j,i-1,j]<=0;

ct23=  forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2==0) )
                   Y[c,i,j,i,j+1]<=X[c,i,j];

ct29=  forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2==0) )
                   Y[c,i,j,i,j+1]<=X[c,i,j+1];

forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2==0) )
                   -Y[c,i,j,i,j+1]<=0;
ct24=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2==0) )
                   Y[c,i,j,i+1,j+1]<=X[c,i,j];


ct30=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2==0) )
                   Y[c,i,j,i+1,j+1]<=X[c,i+1,j+1];

forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2==0) )
                   -Y[c,i,j,i+1,j+1]<=0;

ct25=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i< n-1 &&
                          j>2 && j<m-1) && (j mod 2==0) )
                   Y[c,i,j,i+1,j]<=X[c,i,j];


ct31=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i< n-1 &&
                          j>2 && j<m-1) && (j mod 2==0) )
                   Y[c,i,j,i+1,j]<=X[c,i+1,j];
```

```
forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i< n-1 &&
                         j>2 && j<m-1) && (j mod 2==0) )
                   -Y[c,i,j,i+1,j]<=0;
ct26=  forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                         j>2 && j<m-1 )&& (j mod 2==0) )
                   Y[c,i,j,i+1,j-1]<=X[c,i,j];


ct32=  forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                         j>2 && j<m-1 )&& (j mod 2==0) )
                   Y[c,i,j,i+1,j-1]<=X[c,i+1,j-1];

forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                         j>2 && j<m-1 )&& (j mod 2==0) )
                   -Y[c,i,j,i+1,j-1]<=0;
ct27=  forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i< n-1 &&
                         j>2 && j<m-1) && (j mod 2==0) )
                   Y[c,i,j,i,j-1]<=X[c,i,j];


ct33=  forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i< n-1 &&
                         j>2 && j<m-1) && (j mod 2==0) )
                   Y[c,i,j,i,j-1]<=X[c,i,j-1];

forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i< n-1 &&
                         j>2 && j<m-1) && (j mod 2==0) )
                   -Y[c,i,j,i,j-1]<=0;

// j is odd
ct34= forall ( c in clusters, i in row, j in column :
                      ( c==1 && i> 2 && i<n-1 &&
                        j>2 && j<m-1 ) && (j mod 2!=0) )
                   Y[c,i,j,i-1,j]<=X[c,i,j];


ct40= forall ( c in clusters, i in row, j in column :
                      ( c==1 && i> 2 && i<n-1 &&
                        j>2 && j<m-1 ) && (j mod 2!=0) )
                   Y[c,i,j,i-1,j]<=X[c,i-1,j];
```

```
forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                         j>2 && j<m-1 ) && (j mod 2!=0) )
                    -Y[c,i,j,i-1,j]<=0;
ct35=  forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    Y[c,i,j,i-1,j+1]<=X[c,i,j];


ct41=  forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    Y[c,i,j,i-1,j+1]<=X[c,i-1,j+1];

forall ( c in clusters, i in row, j in column :
                       ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    -Y[c,i,j,i-1,j+1]<=0;

ct36=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    Y[c,i,j,i,j+1]<=X[c,i,j];


ct42=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    Y[c,i,j,i,j+1]<=X[c,i,j+1];

 forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    -Y[c,i,j,i,j+1]<=0;
ct37=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i>2  && i< n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    Y[c,i,j,i+1,j]<=X[c,i,j];


ct43=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i< n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    Y[c,i,j,i+1,j]<=X[c,i+1,j];

forall ( c in clusters, i in row, j in column :
```

```
                        ( c==1 && i> 2 && i< n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    -Y[c,i,j,i+1,j]<=0;
 ct38=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1 )&& (j mod 2!=0) )
                    Y[c,i,j,i,j-1]<=X[c,i,j];


 ct44=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1 )&& (j mod 2!=0) )
                    Y[c,i,j,i,j-1]<=X[c,i,j-1];

   forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i<n-1 &&
                          j>2 && j<m-1 )&& (j mod 2!=0) )
                    -Y[c,i,j,i,j-1]<=0;
 ct39=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i< n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    Y[c,i,j,i-1,j-1]<=X[c,i,j];


 ct45=  forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i< n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    Y[c,i,j,i-1,j-1]<=X[c,i-1,j-1];

 forall ( c in clusters, i in row, j in column :
                        ( c==1 && i> 2 && i< n-1 &&
                          j>2 && j<m-1) && (j mod 2!=0) )
                    -Y[c,i,j,i-1,j-1]<=0;

//[1] neighbours of  (real) left upper  corner X(2,2)

             forall ( c in clusters, i in row ,j in column :
                  c==1 && i==2 && j==2 )
                  Y[c,i,j,i,j+1]<=X[c,i,j];
             forall ( c in clusters, i in row ,j in column :
                  c==1 && i==2 && j==2 )
                  Y[c,i,j,i,j+1]<=X[c,i,j+1];
             forall ( c in clusters, i in row ,j in column :
                  c==1 && i==2 && j==2 )
                  -Y[c,i,j,i,j+1]<=0;
             forall ( c in clusters, i in row ,j in column :
                  c==1 && i==2 && j==2 )
```

```
                 Y[c,i,j,i+1,j+1]<=X[c,i,j];
            forall ( c in clusters, i in row ,j in column :
                 c==1 && i==2 && j==2 )
                 Y[c,i,j,i+1,j+1]<=X[c,i+1,j+1];
            forall ( c in clusters, i in row ,j in column :
                 c==1 && i==2 && j==2 )
                 -Y[c,i,j,i+1,j+1]<=0;
            forall ( c in clusters, i in row ,j in column :
                 c==1 && i==2 && j==2 )
                 Y[c,i,j,i+1,j]<=X[c,i,j];
            forall ( c in clusters, i in row ,j in column :
                 c==1 && i==2 && j==2 )
                 Y[c,i,j,i+1,j]<=X[c,i+1,j];
            forall ( c in clusters, i in row ,j in column :
                 c==1 && i==2 && j==2 )
                 -Y[c,i,j,i+1,j]<=0;

//[2] neighbour of X(2,1)

            forall ( c in clusters, i in row ,j in column :
                 c==1 && i==2 && j==1 )
                 Y[1,i,j,i,j+1]<= X[1,i,j+1];
            forall ( c in clusters, i in row ,j in column :
                 c==1 && i==2 && j==1 )
                 -Y[1,i,j,i,j+1]<= 0;

//[3] neighbours of X(1,2)

            forall ( c in clusters, i in row, j in column :
                 c==1 && i==1 && j==2   )
                Y[1,i,j,i+1,j] <=X[c,i+1,j];
            forall ( c in clusters, i in row, j in column :
                 c==1 && i==1 && j==2   )
                -Y[1,i,j,i+1,j] <=0;
            forall ( c in clusters, i in row, j in column :
                 c==1 && i==1 && j==2   )
                Y[1,i,j,i+1,j+1] <=X[c,i+1,j+1];
            forall ( c in clusters, i in row, j in column :
                 c==1 && i==1 && j==2   )
                -Y[1,i,j,i+1,j+1] <=0;

//[4] neighbours of X(2,m-1)  (real) right upper corner (m is even)

            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j==m-1  && (m mod 2==0) )
                Y[1,i,j,i+1,j]<=X[c,i,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j==m-1  && (m mod 2==0) )
                Y[1,i,j,i+1,j]<=X[c,i+1,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j==m-1  && (m mod 2==0) )
                -Y[1,i,j,i+1,j]<=0;
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j==m-1  && (m mod 2==0) )
                Y[1,i,j,i,j-1]<=X[c,i,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j==m-1  && (m mod 2==0) )
                Y[1,i,j,i,j-1]<=X[c,i,j-1];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j==m-1  && (m mod 2==0) )
                -Y[1,i,j,i,j-1]<=0;

//[5] neighbour of X(1,m) (m is  even)

             forall( c in clusters, i in row, j in column :
                c==1 && i==1 && j==m && (m mod 2==0) )
                Y[1,i,j,i+1,j-1]<=X[1,i+1,j-1];
             forall( c in clusters, i in row, j in column :
                c==1 && i==1 && j==m && (m mod 2==0) )
                -Y[1,i,j,i+1,j-1]<=0;

//[6] neighbours of X(1,m-1) (m is even)

            forall( c in clusters, i in row, j in column :
                c==1 && i==1 && j==m-1 && (m mod 2==0) )
                Y[1,i,j,i+1,j]<=X[1,i+1,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==1 && j==m-1 && (m mod 2==0) )
                -Y[1,i,j,i+1,j]<=0;

//[7] neighbours of right upper corner X(2,m-1)  (m is odd)
           forall( c in clusters, i in row, j in column :
                 c==1 && i==2 && j==m-1 && (m mod 2!=0)  )
                Y[1,i,j,i+1,j]<=X[c,i,j];
           forall( c in clusters, i in row, j in column :
                 c==1 && i==2 && j==m-1 && (m mod 2!=0)  )
                Y[1,i,j,i+1,j]<=X[c,i+1,j];
           forall( c in clusters, i in row, j in column :
                 c==1 && i==2 && j==m-1 && (m mod 2!=0)  )
                -Y[1,i,j,i+1,j]<=0;
           forall( c in clusters, i in row, j in column :
                 c==1 && i==2 && j==m-1 && (m mod 2!=0)  )
                Y[1,i,j,i+1,j-1]<=X[c,i,j];
           forall( c in clusters, i in row, j in column :
                 c==1 && i==2 && j==m-1 && (m mod 2!=0)  )
                Y[1,i,j,i+1,j-1]<=X[c,i+1,j-1];
           forall( c in clusters, i in row, j in column :
                 c==1 && i==2 && j==m-1 && (m mod 2!=0)  )
                -Y[1,i,j,i+1,j-1]<=0;
           forall( c in clusters, i in row, j in column :
                 c==1 && i==2 && j==m-1 && (m mod 2!=0)  )
                Y[1,i,j,i,j-1]<=X[c,i,j];
           forall( c in clusters, i in row, j in column :
                 c==1 && i==2 && j==m-1 && (m mod 2!=0)  )
                Y[1,i,j,i,j-1]<=X[c,i,j-1];
           forall( c in clusters, i in row, j in column :
                 c==1 && i==2 && j==m-1 && (m mod 2!=0)  )
                -Y[1,i,j,i,j-1]<=0;

//[8] neighbours of X(1,m-1) (m is odd)

            forall ( c in clusters, i in row, j in column :
                c==1 && i==1 && j==m-1 && (m mod 2!=0) )
                Y[1,i,j,i+1,j-1]<=X[1,i+1,j-1];
            forall ( c in clusters, i in row, j in column :
                c==1 && i==1 && j==m-1 && (m mod 2!=0) )
                -Y[1,i,j,i+1,j-1]<=0;
            forall ( c in clusters, i in row, j in column :
                c==1 && i==1 && j==m-1 && (m mod 2!=0) )
                Y[1,i,j,i+1,j]<=X[1,i+1,j];

//[9] neighbour of X(2,m)(m is  odd)

             forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j==m  && (m mod 2!=0) )
                Y[1,i,j,i,j-1]<=X[1,i,j-1];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j==m  && (m mod 2!=0) )
                -Y[1,i,j,i,j-1]<=0;

//[10] neighbours of middle top X(2,j) (j is odd)

            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2!=0))
                Y[1,i,j,i,j+1]<=X[1,i,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2!=0))
                Y[1,i,j,i,j+1]<=X[1,i,j+1];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2!=0))
                -Y[1,i,j,i,j+1]<=0;
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2!=0))
                Y[1,i,j,i+1,j]<=X[1,i,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2!=0))
                Y[1,i,j,i+1,j]<=X[1,i+1,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2!=0))
                -Y[1,i,j,i+1,j]<=0;
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2!=0))
                Y[1,i,j,i,j-1]<=X[1,i,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2!=0))
                Y[1,i,j,i,j-1]<=X[1,i,j-1];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2!=0))
                -Y[1,i,j,i,j-1]<=0;

//[11] neighbour of midlle top boundary X(1,j) ( j is odd)

            forall ( c in clusters, i in row, j in column :
                c==1 && i==1 && j>2 && j<m-1 && (j mod 2!=0))
                Y[1,i,j,i+1,j]<=X[1,i+1,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i==1 && j>2 && j<m-1 && (j mod 2!=0))
                -Y[1,i,j,i+1,j]<=0;

//[12] neighbours of midlle top X(2,j) (j is even)

            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i,j-1]<=X[1,i,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i,j-1]<=X[1,i,j-1];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                -Y[1,i,j,i,j-1]<=0;
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i,j+1]<=X[1,i,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i,j+1]<=X[1,i,j+1];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                -Y[1,i,j,i,j+1]<=0;
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i+1,j+1]<=X[1,i,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i+1,j+1]<=X[1,i+1,j+1];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                -Y[1,i,j,i+1,j+1]<=0;
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i+1,j]<=X[1,i,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i+1,j]<=X[1,i+1,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                -Y[1,i,j,i+1,j]<=0;
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i+1,j-1]<=X[1,i,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i+1,j-1]<=X[1,i+1,j-1];
            forall( c in clusters, i in row, j in column :
                c==1 && i==2 && j>2 && j<m-1 && (j mod 2==0))
                -Y[1,i,j,i+1,j-1]<=0;

//[13]  neighbours of midlle top boundary X(1,j) (j is even)

            forall( c in clusters, i in row, j in column :
                c==1 && i==1 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i+1,j-1]<=X[1,i+1,j-1];
            forall( c in clusters, i in row, j in column :
                c==1 && i==1 && j>2 && j<m-1 && (j mod 2==0))
                -Y[1,i,j,i+1,j-1]<=0;
            forall( c in clusters, i in row, j in column :
                c==1 && i==1 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i+1,j]<=X[1,i+1,j];
            forall( c in clusters, i in row, j in column :
                c==1 && i==1 && j>2 && j<m-1 && (j mod 2==0))
                -Y[1,i,j,i+1,j]<=0;
            forall( c in clusters, i in row, j in column :
                c==1 && i==1 && j>2 && j<m-1 && (j mod 2==0))
                Y[1,i,j,i+1,j+1]<=X[1,i+1,j+1];
            forall( c in clusters, i in row, j in column :
```

```
            c==1 && i==1 && j>2 && j<m-1 && (j mod 2==0))
            -Y[1,i,j,i+1,j+1]<=0;

//[14] neighbours of left  X(i,2)

        forall( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            Y[1,i,j,i-1,j]<=X[1,i,j];
        forall( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            Y[1,i,j,i-1,j]<=X[1,i-1,j];
        forall( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            -Y[1,i,j,i-1,j]<=0;
        forall( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            Y[1,i,j,i,j+1]<=X[1,i,j];
        forall( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            Y[1,i,j,i,j+1]<=X[1,i,j+1];
        forall( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            -Y[1,i,j,i,j+1]<=0;
        forall( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            Y[1,i,j,i+1,j+1]<=X[1,i,j];
        forall( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            Y[1,i,j,i+1,j+1]<=X[1,i+1,j+1];
        forall( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            -Y[1,i,j,i+1,j+1]<=0;
        forall ( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            Y[1,i,j,i+1,j]<=X[1,i,j];
        forall ( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            Y[1,i,j,i+1,j]<=X[1,i+1,j];
        forall ( c in clusters, i in row, j in column :
            c==1 && i>2 && i<n-1  && j==2)
            -Y[1,i,j,i+1,j]<=0;

//[15] neighbours left boundary X(i,1)

        forall( c in clusters, i in row, j in column :
            c==1 && i>2 && i<=n-1  && j==1)
            Y[1,i,j,i-1,j+1]<=X[1,i-1,j+1];
            forall( c in clusters, i in row, j in column :
                c==1 && i>2 && i<=n-1  && j==1)
                -Y[1,i,j,i-1,j+1]<=0;
            forall( c in clusters, i in row, j in column :
                c==1 && i>2 && i<=n-1  && j==1)
                Y[1,i,j,i,j+1]<=X[1,i,j+1] ;
            forall( c in clusters, i in row, j in column :
                c==1 && i>2 && i<=n-1  && j==1)
                -Y[1,i,j,i,j+1]<=0 ;

//[16] neighbours of right  X(i,m-1) (m is even)

            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                Y[1,i,j,i-1,j]<=X[1,i,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                Y[1,i,j,i-1,j]<=X[1,i-1,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                -Y[1,i,j,i-1,j]<=0;
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                Y[1,i,j,i-1,j-1]<=X[1,i,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                Y[1,i,j,i-1,j-1]<=X[1,i-1,j-1];
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                -Y[1,i,j,i-1,j-1]<=0;
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                Y[1,i,j,i,j-1]<=X[1,i,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                Y[1,i,j,i,j-1]<=X[1,i,j-1];
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                -Y[1,i,j,i,j-1]<=0;
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                Y[1,i,j,i+1,j]<=X[1,i,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                Y[1,i,j,i+1,j]<=X[1,i+1,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2==0))
                -Y[1,i,j,i+1,j]<=0;

//[17] neighbours of  right boundary  X(i,m) ( m is even)

             forall ( c in clusters, i in row, j in column :
                 c==1 && i>=2 && i<n-1 &&  j==m && (m mod 2==0))
                 Y[1,i,j,i,j-1]<=X[1,i,j-1];
             forall ( c in clusters, i in row, j in column :
                 c==1 && i>=2 && i<n-1 &&  j==m && (m mod 2==0))
                 -Y[1,i,j,i,j-1]<=0;
             forall ( c in clusters, i in row, j in column :
                 c==1 && i>=2 && i<n-1 &&  j==m && (m mod 2==0))
                 Y[1,i,j,i+1,j-1]<=X[1,i+1,j-1];
             forall ( c in clusters, i in row, j in column :
                 c==1 && i>=2 && i<n-1 &&  j==m && (m mod 2==0))
                 -Y[1,i,j,i+1,j-1]<=0;

//[18] neighbours of right X(i,m-1) (m is odd)

             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 Y[1,i,j,i-1,j]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 Y[1,i,j,i-1,j]<=X[1,i-1,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 -Y[1,i,j,i-1,j]<=0;
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 Y[1,i,j,i,j-1]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 Y[1,i,j,i,j-1]<=X[1,i,j-1];
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 -Y[1,i,j,i,j-1]<=0;
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 Y[1,i,j,i+1,j-1]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 Y[1,i,j,i+1,j-1]<=X[1,i+1,j-1];
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 -Y[1,i,j,i+1,j-1]<=0;
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 Y[1,i,j,i+1,j]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 Y[1,i,j,i+1,j]<=X[1,i+1,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<n-1 &&  j==m-1 && (m mod 2!=0))
                 -Y[1,i,j,i+1,j]<=0;

//[19] neighbours of  right boundary X(i,m) (m is odd)

             forall ( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<=n-1 &&  j==m && (m mod 2!=0))
                 Y[1,i,j,i-1,j-1]<=X[1,i-1,j-1];
             forall ( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<=n-1 &&  j==m && (m mod 2!=0))
                 -Y[1,i,j,i-1,j-1]<=0;
             forall ( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<=n-1 &&  j==m && (m mod 2!=0))
                 Y[1,i,j,i,j-1]<=X[1,i,j-1];
             forall ( c in clusters, i in row, j in column :
                 c==1 && i>2 && i<=n-1 &&  j==m && (m mod 2!=0))
                 -Y[1,i,j,i,j-1]<=0;
//[20] neighbours of  bottom left corner X(n-1,2)

            forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==2 )
                 Y[1,i,j,i-1,j]<=X[1,i,j];
            forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==2 )
                 Y[1,i,j,i-1,j]<=X[1,i-1,j];
            forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==2 )
                 -Y[1,i,j,i-1,j]<=0;
            forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==2 )
                 Y[1,i,j,i,j+1]<=X[1,i,j];
            forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==2 )
                 Y[1,i,j,i,j+1]<=X[1,i,j+1];
            forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==2 )
                 -Y[1,i,j,i,j+1]<=0;

//[21] neighbour of  bottom left boundary  X(n,1)

             forall ( c in clusters, i in row, j in column :
                 c==1 && i==n && j==1 )
                  Y[1,i,j,i-1,j+1]<=X[1,i-1,j+1];
             forall ( c in clusters, i in row, j in column :
                 c==1 && i==n && j==1 )
                  -Y[1,i,j,i-1,j+1]<=0;

//[22] neighbour of bottom left boundary  X(n,2)

             forall( c in clusters, i in row, j in column :
                 c==1 && i==n && j==2 )
                 Y[1,i,j,i-1,j]<=X[1,i-1,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i==n && j==2 )
                 -Y[1,i,j,i-1,j]<=0;

//[23] neighbours of bottom right corner X(n-1,m-1) (m is even)

             forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2==0) )
                  Y[1,i,j,i-1,j]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2==0) )
                  Y[1,i,j,i-1,j]<=X[1,i-1,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2==0) )
                  -Y[1,i,j,i-1,j]<=0;
             forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2==0) )
                  Y[1,i,j,i-1,j-1]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2==0) )
                  Y[1,i,j,i-1,j-1]<=X[1,i-1,j-1];
             forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2==0) )
                  -Y[1,i,j,i-1,j-1]<=0;
             forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2==0) )
                  Y[1,i,j,i,j-1]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2==0) )
                  Y[1,i,j,i,j-1]<=X[1,i,j-1];
             forall( c in clusters, i in row, j in column :
                 c==1 && i==n-1 && j==m-1 && (m mod 2==0) )
                  -Y[1,i,j,i,j-1]<=0;

//[24] neighbour of bottom right boundary X(n-1,m) (m is even)
```

```
            forall ( c in clusters, i in row, j in column :
                c==1 && i==n-1 && j==m && (m mod 2==0) )
                 Y[1,i,j,i,j-1]<=X[1,i,j-1];
            forall ( c in clusters, i in row, j in column :
                c==1 && i==n-1 && j==m && (m mod 2==0) )
                 -Y[1,i,j,i,j-1]<=0;

//[25] neighbour of bottom right boundary  X(n,m-1) (m is even)

            forall ( c in clusters, i in row, j in column :
                c==1 && i==n && j==m-1 && (m mod 2==0) )
                 Y[1,i,j,i-1,j]<=X[1,i-1,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i==n && j==m-1 && (m mod 2==0) )
                 -Y[1,i,j,i-1,j]<=0;
            forall ( c in clusters, i in row, j in column :
                c==1 && i==n && j==m-1 && (m mod 2==0) )
                 Y[1,i,j,i-1,j-1]<=X[1,i-1,j-1];
            forall ( c in clusters, i in row, j in column :
                c==1 && i==n && j==m-1 && (m mod 2==0) )
                 -Y[1,i,j,i-1,j-1]<=0;

//[26] neighbour of bottom right corner X(n-1,m-1) (m is odd)

            forall ( c in clusters, i in row, j in column :
                c==1 && i==n-1 && j==m-1 && (m mod 2!=0) )
                 Y[1,i,j,i-1,j]<=X[1,i,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i==n-1 && j==m-1 && (m mod 2!=0) )
                 Y[1,i,j,i-1,j]<=X[1,i-1,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i==n-1 && j==m-1 && (m mod 2!=0) )
                 -Y[1,i,j,i-1,j]<=0;
            forall ( c in clusters, i in row, j in column :
                c==1 && i==n-1 && j==m-1 && (m mod 2!=0) )
                 Y[1,i,j,i,j-1]<=X[1,i,j];
            forall ( c in clusters, i in row, j in column :
                c==1 && i==n-1 && j==m-1 && (m mod 2!=0) )
                 Y[1,i,j,i,j-1]<=X[1,i,j-1];
            forall ( c in clusters, i in row, j in column :
                c==1 && i==n-1 && j==m-1 && (m mod 2!=0) )
                 -Y[1,i,j,i,j-1]<=0;

//[27] neighbour of bottom right boundary X(n,m)(m is odd)

            forall ( c in clusters, i in row, j in column :
                c==1 && i==n && j==m && (m mod 2!=0) )
```

```
                  Y[1,i,j,i-1,j-1]<=X[1,i-1,j-1];
             forall ( c in clusters, i in row, j in column :
                 c==1 && i==n && j==m && (m mod 2!=0) )
                  -Y[1,i,j,i-1,j-1]<=0;

//[28] neighbour of bottom right boundary X(n,m-1)(m is odd)

            forall ( c in clusters, i in row, j in column :
                 c==1 && i==n && j==m-1 && (m mod 2!=0) )
                  Y[1,i,j,i-1,j]<=X[1,i-1,j];
            forall ( c in clusters, i in row, j in column :
                 c==1 && i==n && j==m-1 && (m mod 2!=0) )
                  -Y[1,i,j,i-1,j]<=0;

//[29] neighbours of bottom X(n-1,j)(j is odd)

             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i,j-1]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i,j-1]<=X[1,i,j-1];
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 -Y[1,i,j,i,j-1]<=0;
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i-1,j-1]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i-1,j-1]<=X[1,i-1,j-1];
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 -Y[1,i,j,i-1,j-1]<=0;
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i-1,j]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i-1,j]<=X[1,i-1,j];
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 -Y[1,i,j,i-1,j]<=0;
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i-1,j+1]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i-1,j+1]<=X[1,i-1,j+1];
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 -Y[1,i,j,i-1,j+1]<=0;
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i,j+1]<=X[1,i,j];
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i,j+1]<=X[1,i,j+1];
             forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2!=0))
                 -Y[1,i,j,i,j+1]<=0;

//[30]neighbours of bottom boundary X(n,j) ( j is odd)

             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i-1,j-1]<=X[1,i-1,j-1];
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n && j> 2 && j< (m-1)  && (j mod 2!=0))
                 -Y[1,i,j,i-1,j-1]<=0;
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i-1,j]<=X[1,i-1,j];
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n && j> 2 && j< (m-1)  && (j mod 2!=0))
                 -Y[1,i,j,i-1,j]<=0;
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n && j> 2 && j< (m-1)  && (j mod 2!=0))
                 Y[1,i,j,i-1,j+1]<=X[1,i-1,j+1];
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n && j> 2 && j< (m-1)  && (j mod 2!=0))
                 -Y[1,i,j,i-1,j+1]<=0;

//[31] neighbours of bottom X(n-1,j) (j is even)

             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2==0))
                 Y[1,i,j,i,j-1]<=X[1,i,j];
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2==0))
                 Y[1,i,j,i,j-1]<=X[1,i,j-1];
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2==0))
                 -Y[1,i,j,i,j-1]<=0;
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2==0))
                 Y[1,i,j,i-1,j]<=X[1,i,j];
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2==0))
                 Y[1,i,j,i-1,j]<=X[1,i-1,j];
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2==0))
                 -Y[1,i,j,i-1,j]<=0;
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2==0))
                 Y[1,i,j,i,j+1]<=X[1,i,j];
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2==0))
                 Y[1,i,j,i,j+1]<=X[1,i,j+1];
             forall ( c in clusters, i in row, j in column :
                 c==1 &&  i==n-1 && j> 2 && j< (m-1)  && (j mod 2==0))
                 -Y[1,i,j,i,j+1]<=0;

//[32] neighbour of bottom boundary X(n,j)(j is even)

            forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n && j> 2 && j< (m-1)  && (j mod 2==0))
                 Y[1,i,j,i-1,j]<=X[1,i-1,j];
            forall( c in clusters, i in row, j in column :
                 c==1 &&  i==n && j> 2 && j< (m-1)  && (j mod 2==0))
                 -Y[1,i,j,i-1,j]<=0;

 //-----------------------------------------------------------------------------------------;
  // From the distance part
   // j is odd
   // j<=l, i<=k
    ct46= forall  (c in clusters, i in row,k in row ,j in column,l in column:
                   (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i<k  && j<l && (j mod 2!=0) )
                    X[c,i,j] + X[c,k,l] -Y[c,i,j,k,l] <=1;

    forall  (c in clusters, i in row,k in row ,j in column,l in column:
                   (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i<k  && j<l && (j mod 2!=0) )
                     -Y[c,i,j,k,l] <=0;

    forall (c in clusters, i in row,k in row ,j in column,l in column:
                   (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  ((i==k  && j<l) || (i<k  && j==l) ) && (j mod 2!=0)  )
                   X[c,i,j] + X[c,k,l] -Y[c,i,j,k,l] <=1;
    forall (c in clusters, i in row,k in row ,j in column,l in column:
                   (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  ((i==k  && j<l) || (i<k  && j==l) ) && (j mod 2!=0)  )
                    -Y[c,i,j,k,l] <=0;
   // j<=l, i> k
    ct47= forall (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i>k  &&  j<=l  && (j mod 2!=0))
                    X[c,i,j] + X[c,k,l] -Y[c,i,j,k,l] <=1;
    forall (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i>k  &&  j<=l  && (j mod 2!=0))
                    -Y[c,i,j,k,l] <=0;
  // j is even
    // j<=l, i<=k
    ct48= forall (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i<k  && j<l && (j mod 2 ==0) )
                    X[c,i,j] + X[c,k,l] -Y[c,i,j,k,l] <=1;
    forall (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i<k  && j<l && (j mod 2 ==0) )
                    -Y[c,i,j,k,l] <=0;
  forall(c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  ((i==k  && j<l) || (i<k  && j==l) ) && (j mod 2 ==0)  )
                    X[c,i,j] + X[c,k,l] -Y[c,i,j,k,l] <=1;
  forall(c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  ((i==k  && j<l) || (i<k  && j==l) ) && (j mod 2 ==0)  )
                    -Y[c,i,j,k,l] <=0;
   // j<=l, i>k
   ct49= forall  (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i>k  &&  j<=l  && (j mod 2==0))
                    X[c,i,j] + X[c,k,l] -Y[c,i,j,k,l] <=1;
forall  (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i>k  &&  j<=l  && (j mod 2==0))
                    -Y[c,i,j,k,l] <=0;
  //l is odd =
  // j>l, i<=k
   ct50= forall (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i<=k  && j>l  && (l mod 2 !=0))
                   X[c,i,j] + X[c,k,l] -Y[c,i,j,k,l] <=1;
 forall (c in clusters, i in row,k in row ,j in column,l in column:
```

```
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i<=k  && j>l  && (l mod 2 !=0))
                    -Y[c,i,j,k,l] <=0;
   // j>l, i > k
  ct51= forall (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i>k  &&  j>l   && (l mod 2!=0))
                    X[c,i,j] + X[c,k,l] -Y[c,i,j,k,l] <=1;
   forall (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i>k  &&  j>l   && (l mod 2!=0))
                     -Y[c,i,j,k,l] <=0;
   // l is even =
   // j>l, i<=k
   ct52= forall (c in clusters, i in row,k in row ,j in column,l in column:
                 (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    && i<=k && j>l && (l mod 2==0))
                 X[c,i,j] + X[c,k,l] -Y[c,i,j,k,l] <=1;
   forall (c in clusters, i in row,k in row ,j in column,l in column:
                 (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    && i<=k && j>l && (l mod 2==0))
                  -Y[c,i,j,k,l] <=0;
   // j>l, i>k
    ct53= forall (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i>k  &&  j>l  && (l mod 2==0) )
                   X[c,i,j] + X[c,k,l] -Y[c,i,j,k,l] <=1;
   forall (c in clusters, i in row,k in row ,j in column,l in column:
                    (c!=1 && i!=1 && i!=n && j!=1&& j!=m && k!=1 && k!=n && l!=1&& l!=m)
                    &&  i>k  &&  j>l  && (l mod 2==0) )
                    -Y[c,i,j,k,l] <=0;

}

execute{
 for(var counter1 in clusters)
 for(var counter2 in row)
 for(var counter3 in column)
 x_temp[(counter1-1)*n+counter2][counter3]=X[counter1][counter2][counter3];
 //area_temp[counter][counter2][counter3]=areaofs[(counter-1)*n+counter2][counter3];
  range1 = "Sheet5!A1:W"+n*c;
  obj_final = cplex.getObjValue();
 // thisOplModel.postProcess() ;
 }

main{
thisOplModel.generate();
```

```
cplex.exportModel("hexogan_new.lp");
cplex.solve();
thisOplModel.postProcess() ;
writeln("Objective value = ",cplex.getObjValue());
writeln("Best Objective value = ",cplex.getBestObjValue());
writeln("CPLEX status = ",cplex.getCplexStatus());
}
```

Bibliography

[1] http://xgandibleux.free.fr/MOCOlib/MOSCP.html.

[2] http://www.ilog.com.

[3] R. K. Ahuja, O. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.

[4] E. Angel, E. Bampis, and L. Gourves. Approximating the Pareto curve with local search for the bicriteria TSP (1,2) problem. *Theoretical Computer Science*, 310:135–146, 2004.

[5] E. Angel, E. Bampis, L. Gourves, and J. Monnot. (non)-approximability for the multicriteria tsp (1,2). *Fundamentals of Computation Theory 15th International Symposium, Lecture Notes in Computer Science*, 3623:329–340, 2005.

[6] R. Armann. Solving multiobjective programming problems by discrete representation. *Optimization*, 20(4):483–492, 2000.

[7] C. Bazgan, H. Hugot, and D. Vanderpooten. Implementing an efficient fptas for the 0–1 multi-objective knapsack problem. *European Journal of Operational Research*, 198(1):47–56, 2009.

[8] F. A. Behringer. Lexicographic quasiconcave multiobjective programming. *Zeitschrift fur Operations Research*, 21:103–116, 1977.

[9] P. Belotti, B. Soylu, and M. M. Wiecek. *A branch-and-bound algorithm for biobjective mixed-integer programs.* Technical Report, Department of Mathematical Sciences, Clemson University, SC, 2012.

[10] V. Blouin, B. J. Hunt, and M. M. Wiecek. *MCDM with relative importance of criteria: application to configuration design of vehicles.* T. Trzaskalik, T. Wachowicz, editors, the Karol Adamiecki University of Economics in Katowice, Katowice, Poland Verlag, 2009.

[11] P. A. N. Bosman and D. Thierens. The balance between proximity and diversity in multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(2):174–188, 2003.

[12] V. J. Bowman. *On the relationship of the Chebyshev norm and the efficient frontier of multiple-criteria objectives.* Lecture Notes in Economics and Mathematical Systems 130, 1975.

[13] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47:730–743, 1999.

[14] L. G. Chalmet, L. Lemonidis, and D. J. Elzinga. An algorithm for the bi-criterion integer programming problem. *European Journal of Operational Research*, 25:292–300, 1986.

[15] T. Chvátal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[16] M. A. Clemens, C. S. ReVelle, and J. C. Williams. Reserve design for species preservation. *European Journal of Operational Research*, 112:273–283, 1999.

[17] C. C. Coello, V. D. Van, and G. Lamont. *Evolutionary algorithms for solving multi-objective problems*, volume 46. Kluwer Academic Publishers, 2002.

[18] A. Colorni, M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian. Heuristics from nature for hard combinatorial optimization problems. *International Transactions in Operational Research*, 3(1):1–21, 1996.

[19] B. Csuti, S. Polasky, P. H. Williams, R. L. Pressey, J. D. Camm, M. Kershaw, A. R. Kiester, B. Downs, R. Hamilton, M. Huso, and K. Sahr. A comparison of reserve selection algorithms using data on terrestrial vertebrates in oregon. *Biological Conservation*, 80:83–97, 1997.

[20] M. S. Daskin and E. H. Stern. A hierarchical objective set covering model for emergency medical service vehicle deployment. *Operations Research Society of America*, 15(2):137–151, 1981.

[21] K. Deb. *Multi-objective Optimization using evolutionary algorithms,Wiley interscience series in systems and optimization.* John Wiley and Sons Ltd, 2001.

[22] I. Diakonikolas and M. Yannakakis. Small approximate Pareto sets for bi-objective shortest paths and other problems. *SIAM Journal on Computing*, 39(4):1340–1371, 2001.

[23] R. V. Efremov and G. K. Kamenev. Properties of a method for polyhedral approximation of the feasible criterion set in convex multiobjective problems. *Annals of Operations Research*, 166:271–279, 2009.

[24] M. Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *International Transactions in Operational Research*, 7:5–31, 2001.

[25] M. Ehrgott and E. A. Galperin. Min-max formulation of the balance number in multiobjective global optimization. *Computers and Mathematics with Applications*, 44:899–907, 2002.

[26] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22:425–460, 2000.

[27] M. Ehrgott and M. M. Wiecek. *Multiple Criteria Decision Analysis: State of the Art Surveys.* Figueira J., Greco S., and Ehrgott M., editors, Springer Verlag, 2005.

[28] A. Engau and M. M. Wiecek. Cone characterizations of approximate solutions in real-vector optimization. *Journal of Optimization Theory and Applications*, 134:499–513, 2007.

[29] M. Ergott. *Multicriteria Optimization.* Springer, Second edition, 2005.

[30] T. Erlebach, H. Kellerer, and U. Pferschy. Approximating the multiobjective knapsack problems. *Management Science*, 48:1603–1612, 2002.

[31] S. L. Faulkenberg and M. M. Wiecek. On the quality of discrete representations in multiple objective programming. *Optimization and Engineering*, 11:423–440, 2010.

[32] C. Filippi and E. Stevanato. Approximation schemes for bi-objective combinatorial optimization and their application to the TSP with profits. *Operations Research*, 40:2418–2428, 2013.

[33] D. T. Fischer and R. L. Church. Clustering and compactness in reserve site selection: An extension of the biodiversity management area selection model. *Forest Science*, 49:555–565, 2003.

[34] D. T. Fischer and R. L. Church. The SITES reserve selection system: A critical review. *Environmental Modeling and Assessment*, 10:215–228, 2005.

[35] R. Frankham, J. D. Ballou, and D. A. Briscoe. *Introduction to conservation genetics.* Cambridge University Press, Second edition, 2010.

[36] E. A. Galperin. Nonscalarized multiobjective global optimization. *Optimization Theory and Applications*, 75(1):69–85, 1992.

[37] M. Gardenghi. *Multiobjective Optimization for Complex Systems.* PhD Thesis, Department of Mathematical Sciences, Clemson University, Clemson, SC, 2009.

[38] M. Gardenghi, T. Goḿez, and M. M. Wiecek. Algebra of efficient sets for multiobjective complex systems. *Journal of Optimization Theory and Applications*, 149(2):385–410, 2011.

[39] M. Gardenghi and M. M. Wiecek. Efficiency for multiobjective multidisciplinary optimization problems with quasiseparable subproblems. *Optimization and Engineering*, 13(2):293–318, 2012.

[40] T. Goel, R. Vaidyanathan, R. T. Haftka, W. Shyy, N. V. Queipo, and K. Tucker. Response surface approximation of Pareto optimal front in multi-objective optimization. *Computer Methods in Applied Mechanics and Engineering*, 196:879–893, 2007.

[41] C. H. Goh and X. Q. Yang. Analytic efficient solution set for multi-criteria quadratic programs. *European Journal of Operational Research*, 92:166–181, 1996.

[42] R. T. Haftka and L. T. Watson. Multidisciplinary design optimization with quasiseparable subsystems. *Optimization and Engineering*, 6:9–20, 2005.

[43] M. P. Hansen and A. Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. *Institute of Mathematical Modeling: Technical Report*, 7, 1998.

[44] M. Hartikainen, K. Miettinen, and M. M. Wiecek. Constructing a Pareto front approximation for decision making. *Mathematical Methods of Operations Research*, 73:209–234, 2011.

[45] M. Hartikainen, K. Miettinen, and M. M. Wiecek. Paint: Pareto front interpolation for nonlinear multiobjective optimization. *Computational Optimization and Application*, 52:845–867, 2012.

[46] S. Helbig and D. Pateva. On several concepts for ϵ-efficiency. *OR Spektrum*, 16(2):179–186, 1994.

[47] J. Hof and C. H. Flather. Accounting for connectivity and spatial correlation in the optimal placement of wildlife habitat. *Ecological Modeling*, 88:143–155, 1996.

[48] B. J. Hunt, V. W. Blouin, and M. M. Wiecek. Relative importance of design criteria: a preference modeling approach. *Journal of Mechanical Design*, 129(9):907–914, 2007.

[49] B. J. Hunt, C. Hughes, and M. M. Wiecek. Relative importance of criteria in multiobjective programming: a cone-based approach. *European Journal of Operational Research*, 207:936–945, 2010.

[50] B. J. Hunt and M. M. Wiecek. *Cones to aid decision making in multicriteria programming, In: Multi-Objective Programming and Goal-Programming.* Tanino T., Tanaka T. and M. Inuiguchi M., editors, Springer Berlin, 2003.

[51] J. Jahn. *Vector Optimization. Theory, applications, and extensions.* Springer-Verlag, Berlin, 2004.

[52] A. Jaszkiewicz. Do multiple-objective metaheuristics deliver on their promises? a computational experiment on the set covering problem. *IEEE Transactions on Evolotionary Computation*, 7:133–143, 2003.

[53] A. Jaszkiewicz. A comparative study of multiple objective metaheuristics on the biobjective set covering problem and the Pareto memetic algorithm. *Annals of Operations Research*, 131:135–158, 2004.

[54] N. Jozefowiez, F. Glover, and M. Laguna. Multi-objective meta-heuristics for the traveling salesman problem with profits. *Journal of Mathematical Modelling and Algorithms*, 7(2):177–195, 2008.

[55] R. K. Kincaid, C. Easterling, and M. Jeske. Computational experiments with heuristics for two nature reserve site selection problems. *Computers and Operations Research*, 35:499–512, 2006.

[56] O. N. Klimova and V. D. Noghin. Using interdependent information on the relative importance of criteria in decision making. *Optimization*, 46(12):2178–2190, 2006.

[57] M. M. Kostreva, W. Ogryczak, and A. Wierzbicki. Equitable aggregations and multiple criteria analysis. *European Journal of Operational Research*, 158(2):362–377, 2004.

[58] S. S. Kutateladze. Convex ϵ-programming. *Soviet Mathematics. Doklady*, 20:391–393, 1979.

[59] M. Laumanns, R. Zenklusen, and O. Maler. Stochastic convergence of random search methods to fixed size Pareto front approximations. *European Journal of Operational Research*, 213(2):414–421, 2011.

[60] J. Legriel, C. L. Cotton, and O. Maler. Approximating the Pareto front of the multi-criteria optimization problems. *19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 69–83, 2010.

[61] Y. H. Liu. A heuristic algorithm for the multicriteria set covering problems. *Applied Mathematics Letters*, 6:21–23, 1993.

[62] P. Loridan. ϵ-solutions in vector minimization problems. *Journal of Optimization Theory and Applications*, 43(2):265–276, 1984.

[63] D. G. Luenberger. *Optimization by Vector Space Methods.* John Wiley and Sons, New York, 1969.

[64] T. Lust, J. Teghem, and D. Tuyttens. Very large-scale neighborhood search for solving multiobjective combinatorial optimization problems. *Evolutionary Multi-Criterion Optimization: 6th International Conference*, 6576:254–268, 2011.

[65] B. Manthey and L. S. Ram. Approximation algorithms for the multicriteria traveling salesman problems. *Algorithmica*, 53:69–88, 2009.

[66] C. R. Margules and R. L. Pressey. Systematic conservation planning. *Nature*, 405:243–253, 2000.

[67] J. Martin, C. Bielza, and D. R. Insua. Approximating nondominated sets in continuous multiobjective optimization problems. *Naval Research Logistics*, 52:469–480, 2005.

[68] M. D. McDonnell, H. P. Possingham, I. R. Ball, and E. A. Cousins. Mathematical methods for spatially cohesive reserve design. *Environmental Modeling and Assessment*, 7:107–114, 2002.

[69] K. Miettinen. *Nonlinear Multiobjective Optimization.* Kluwer Academic Publishers, 1999.

[70] N. Musliu. Local search algorithm for unicost set covering problem. *Advances in Applied Artificial Intelligence*, 4031:302–311, 2006.

[71] D. J. Nalle, J. L. Arthur, C. A. Montgomery, and J. Sessions. Economic and spatial impacts of an existing reserve network on future augmentation. *Environmental Modeling and Assessment*, 7:99–105, 2002.

[72] D. J. Nalle, J. L. Arthur, and J. Sessions. Designing compact and contiguous reserve networks with a hybrid heuristic algorithm. *Forest Science*, 48:59–68, 2002.

[73] V. D. Noghin. Relative importance of criteria: a quantitative approach. *Journal of Multicriteria Decision Analysis*, 6:355–363, 1997.

[74] V. D. Noghin and I. V. Tolstykh. Using quantitative information on the relative importance of criteria for decision making. *Computational Mathematics and Mathematical Physics*, 40(11):1529–1536, 2000.

[75] H. Onal. First-best, second-best, and heuristic solutions in conservation reserve site selection. *Biological Conservation*, 115:55–62, 2003.

[76] H. Onal and R. A. Briers. Incorporating spatial criteria in optimum reserve network selection. *Proceedings of the Royal Society of London, Biological Sciences*, 269:2437–2441, 2002.

[77] H. Onal and R. A. Briers. Selection of a minimum-boundary reserve network using integer programming. *Proceedings of the Royal Society of London, Biological Sciences*, 270:1487–1491, 2003.

[78] H. Onal and R. A. Briers. Designing a conservation reserve network with minimal fragmentation: A linear integer programming approach. *Environmental Modeling and Assessment*, 10:193–202, 2005.

[79] H. Onal and Y. Wang. A graph theory approach for designing conservation reserve networks with minimal fragmentation. *Networks*, 51:142–152, 2008.

[80] J. M. A. Pangilinan and G. K. Janssens. Evolutionary algorithms for the multiobjective shortest path problem. *International Journal of Computer and Information Engineering*, 1:1–5, 2007.

[81] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. *Proceedings 41st Annual Symposium on Foundations of Computer Science*, 1:86–92, 2000.

[82] N. Pareto. *Cours d'Économie Politique*. Rouge, Lausanne, Switzerland, 1896.

[83] S. L. Pimm and J. H. Lawton. Planning for biodiversity. *Science*, 279:2068–2069, 1998.

[84] R. B. Primack. *Essentials of conservation biology*. Sinauer Associates, Fifth edition, 2010.

[85] C. Prins and C. Prodhon. Two-phase method and lagrangian relaxation to solve the biobjective set covering problem. *Annals of Operations Research*, 147:23–41, 2006.

[86] C. R. Reeves. *Modern heuristic techniques for combinatorial problems.* Blackwell Scientific Publications, Second edition, 1993.

[87] H. Reuter. An approximation method for the efficiency set of multiobjective programming problems. *Optimization*, 21:905–911, 1990.

[88] C. S. ReVelle, J. C. Williams, and J. J. Boland. Counterpart models in facility location science and reserve selection science. *Environmental Modeling and Assessment*, 7:71–80, 2002.

[89] M. K. Richard. *Complexity of Computer Computations.* Plenum Press, 1972.

[90] R. T. Rockafellar. *Convex analysis.* Princeton Landmarks in Mathematics. Princeton University Press, First edition, 1970.

[91] R. T. Rockafellar and R. J. B. Wets. *Variational analysis (Grundlehren der Mathematischen Wissenschaften).* Springer, Third edition, 1998.

[92] G. Ruhe and B. Fruhwirth. ϵ-optimality for bicriteria programs and its application to minimum cost flows. *Computing*, 44(1):1340–1371, 1990.

[93] S. Ruzika and M. M. Wiecek. Survey paper: Approximation methods in multiobjective programming. *Journal of Optimization Theory and Applications*, 126:473–571, 2005.

[94] M. H. Safer and J. B. Orlin. Fast approximation schemes for multi-criteria combinatorial optimization, technical report. *MIT Sloan School of Management*, 1:3756–3795, 1995.

[95] Y. Sawaragi, H. Nakayama, and T. Tanino. *Theory of Multiobjective Optimization.* Academic Press, Orlando, 1985.

[96] R. R. Saxena and S. R. Arora. Linearization approach to multiobjective set covering problem. *Optimization*, 43:145–156, 1998.

[97] S. Sayin. Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. *Mathematical Programming*, 87(3,Ser. A):543–560, 2000.

[98] T. Shirabe. A model of contiguity for spatial unit allocation. *Geographical Analysis*, 37:2–16, 2005.

[99] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation, and Application.* John Wiley, New York, 1896.

[100] E. L. Ulungu and J. Teghem. Multiobjective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3:83–104, 1994.

[101] S. Vassilvitskii and M. Yannakakis. Efficiently computing succinct trade-off curves. *Theoretical Computer Science*, 348(2-3):334–356, 2005.

[102] V. V. Vazirani. *Approximation Algorithms.* Springer, Second edition, 2003.

[103] D. J. White. Epsilon efficiency. *Journal of Optimization Theory and Applications*, 49(2):319–337, 1986.

[104] M. M. Wiecek. Advances in cone-based preference modeling for decision making with multiple criteria. *Decision Making in Manufacturing and Services*, 1(1-2):153–173, 2007.

[105] J. C. Williams. Optimal reserve site selection with distance requirements. *Computers and Operations Research*, 35:488–498, 2008.

[106] J. C. Williams, C. S. ReVelle, and Simon A. Levin. Spatial attributes and reserve design models: A review. *Environmental Modeling and Assessment*, 10:163–181, 2005.

[107] R. Woodroffe and J. R. Ginsberg. Edge effects and the extinction of populations inside protected areas. *Science*, 280:2126–2128, 1998.

[108] X. Wu and A. T. Murray. A new approach to quantifying spatial contiguity using graph theory and spatial interaction. *International Journal of Geographical Information Science*, 22:387–407, 2008.

[109] X. Wu, A. T. Murray, and N. Xiao. A multiobjective evolutionary algorithm for optimizing spatial contiguity in reserve network design. *Landscape Ecology*, 26:425–437, 2011.

[110] P. L. Yu. Cone convexity, cone extreme points, and ondominated solutions in decision problems with multiobjectives. *Journal of Optimization Theory and Applications*, 14(3):319–377, 1974.

[111] M. Zeleny. *Compromise programming.* J. Cochrane and M. Zeleny, editors, University of South Carolina Press, Columbia, 1973.

[112] E. Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications.* Ph.D. dissertation, Computer Engineering and Networks Laboratory (TIK), ETH, Zurich, Switzerland, 1999.