Chapter 1

Introduction
In this chapter, we present an overview of the basics of optimization, fuzzy optimization, and component-based software development and discuss the structure of the thesis.

1.1 Optimization

Mathematical optimization (alternatively, optimization or mathematical programming) constitutes an important branch of modern applied mathematics. It deals with the task of decision making, which entails choosing among various alternatives. The measure of goodness of the alternatives is described in terms of maximization or minimization of an objective function (performance index or mathematical function). Thus, an optimization framework deals with selecting the best alternative in sense of the given objective function, subject to the constraints for the problem to be solved.

Optimization is central to any problem involving decision making, whether in engineering, operations research, management science, financial engineering or economics. By virtue of its great utility in such diverse areas, optimization holds an important place in the practical, scientific world. In recent years, the area of optimization has received enormous attention, primarily because of the rapid progress in computer technology, including the development and availability of user-friendly software, high-speed and parallel processors, and artificial neural networks. A clear example of this phenomenon is the wide accessibility of optimization software tools, such as Optimization Toolbox of MATLAB, Mathematica, Maple, ILOG, OPL Development Studio, XPRESS-IVE, AIMMS, AMPL, and GAMS. An optimization problem may be classified either as a single-objective optimization problem or as a multiobjective optimization problem.

1.1.1 Single-objective optimization problem

A single-objective optimization problem, finds a maximizer or minimizer of a given objective function, subject to a given set of constraints that must be satisfied by any solution. Mathematically, it can be written in the following form:
The mathematical programming problem $P(1.1)$ is called an LPP if the objective function $f$ is linear and the set $\Omega$ is described by using linear inequalities or equations.

2. The mathematical programming problem $P(1.1)$ is called an NLPP if the objective function $f$ is nonlinear and/or the set $\Omega$ is described by using at least one nonlinear inequality or equation.

*Linear programming* is a fundamental optimization problem and is one of the most widely used decision making tools for solving real-world problems. LPP
has various applications including optimum production/allocation of resources, production scheduling, and diet planning; transportation problems including transshipment problems, minimum cost flows, maximum flows, and shortest path problems; and workforce planning, which consists of optimal assignment of jobs, scheduling of classes, and so on. The problem of solving a system of linear inequalities dates back at least as far as Fourier, after whom the method of Fourier-Motzkin elimination is named. The earliest linear programming was developed by Leonid Kantorovich [56] in 1939 for use during World War II to plan expenditures and returns to reduce costs to the army and increase losses to the enemy. Postwar, many industries found its use in their daily planning. The simplex method developed by George B. Dantzig [28] is an iterative process that provides solutions of LPPs in a finite number of steps. It provides an algorithm that consists of moving from one vertex of the feasible region to another until the optimum or an unbounded solution is found. To overcome the burden of the tedious computations involved in the simplex method, one can solve LPPs with a large number (up to millions) of variables and constraints, using many reliable softwares such as LINDO, CPLEX, IMSL, MINOS, MPSX, and XPRESS-MP. LPPs were first shown to be solvable in polynomial time by Leonid Khachiyan [61] in 1979, but a larger theoretical and practical breakthrough in the field came in 1984 when Narendra Karmarkar [57] introduced a new interior point method for solving LPPs.

John Von Neumann [87] in 1947 introduced in linear programming the concept of duality theory, which was subsequently formulated in a precise form by Gale et al. [33]. The idea of duality theory is to associate with a given LPP called primal problem another LPP called dual problem. Apart from theoretical interest, duality is significant from computational aspects as well. Sometimes, the dual problem has a nicer structure than does the primal problem, and so it might be easier to work with. Furthermore, the dual problem provides bounds on the achievable values of the primal problem.

Consider the primal LPP defined as follows:

\[
P(1.2) \quad \text{max } \sum_{j=1}^{n} c_j x_j \\
\text{subject to } \\
\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, 2, \ldots, m',
\]
\[ \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \; i = m' + 1, m' + 2, \ldots, m'', \]
\[ \sum_{j=1}^{n} a_{ij} x_j = b_i, \; i = m'' + 1, m'' + 2, \ldots, m, \]
\[ x_j \geq 0, \; j = 1, 2, \ldots, n. \]

Let \( y_i \) be the dual variable associated with the \( i \)-th constraint. The dual problem of primal problem \( P(1.2) \) is then defined as follows:

\[
P(1.3) \quad \min \sum_{i=1}^{m} b_i y_i
\]
subject to
\[ \sum_{i=1}^{m} a_{ij} y_i \geq c_j, \; j = 1, 2, \ldots, n, \]
\[ y_i \geq 0, \; i = 1, 2, \ldots, m', \]
\[ y_i \leq 0, \; i = m' + 1, m' + 2, \ldots, m'', \]
\[ y_i \text{ unrestricted in sign, } i = m'' + 1, m'' + 2, \ldots, m. \]

There are two fundamental theorems to duality theory.

**Theorem 1.1. (Weak duality)** For any pair of primal and dual feasible solutions \( x = (x_1, x_2, \ldots, x_n)^T \) and \( y = (y_1, y_2, \ldots, y_m)^T \), respectively,
\[
\sum_{j=1}^{n} c_j x_j \leq \sum_{i=1}^{m} b_i y_i \quad (c^T x \leq b^T y).
\]

**Corollary 1.1.** Let \( \mathbf{x} = (x_1, x_2, \ldots, x_n)^T \) and \( \mathbf{y} = (y_1, y_2, \ldots, y_m)^T \) be any pair of primal and dual feasible solutions, respectively, such that,
\[
\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i \quad (c^T \mathbf{x} = b^T \mathbf{y}),
\]
then \( \mathbf{x} \) is optimal solution to the primal problem and \( \mathbf{y} \) is optimal solution to the dual problem.

**Theorem 1.2. (Strong duality)** If problem \( P(1.2) \) has an optimal solution \( \mathbf{x} \), then problem \( P(1.3) \) has an optimal solution \( \mathbf{y} \) and their optimal objective values are equal, that is,
\[ \sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i \quad (c^T x = b^T y). \]

Note that it is possible to obtain an optimal solution to the dual problem when only an optimal solution to the primal problem is known or vice-versa using the following complementary slackness conditions.

**Theorem 1.3. (Complementary slackness conditions)** For a primal-dual pair of feasible solutions \( \bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)^T \) and \( \bar{y} = (\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_m)^T \) the following conditions are equivalent,

1. both \( \bar{x} \) and \( \bar{y} \) are optimal solutions,
2. \( c^T \bar{x} = b^T \bar{y} \),
3. \( \sum_{i=1}^{m} \bar{y}_i \left( \sum_{j=1}^{n} a_{ij} \bar{x}_j - b_i \right) = 0 \) and \( \sum_{j=1}^{n} \bar{x}_j \left( c_j - \sum_{i=1}^{m} a_{ij} \bar{y}_i \right) = 0 \).

Furthermore, according to duality theory, if the primal problem is unbounded, then the dual problem is infeasible. Also, if the dual problem is unbounded, then the primal problem must be infeasible; however, it is possible for both the dual problem and the primal problem to be infeasible. One of the most important impacts of the duality theory presented above has been on computational procedures used for solving LPPs. First, it has been established that the dual problem can be solved in place of the primal problem whenever there are advantages in doing so. For example, if the number of constraints of a given (primal) problem is much more than the number of variables, it is usually wise to solve the dual problem instead of the primal problem because the solution time increases much more rapidly with the number of constraints in the problem than with the number of variables. Second, new algorithms that take advantage of the duality theory in more subtle ways have been developed.

During mathematical formulation of real-world problems, the value assigned to a parameter is often a rough estimate because gathering accurate data is frequently difficult. Because of the uncertainty about the true value of the parameter, it is important to analyze how the solution derived from a model would change (if at all) when the value assigned to the parameter will be changed to other plausible values. This process is referred to as sensitivity analysis.
In practical situations, we often come across optimization problems that cannot be adequately expressed as LPPs owing to the nonlinear nature of the functions involved. Such programming problems are termed as NLPPs. The beginning of the mathematical theory of nonlinear programming can be dated back to the work done by Kuhn and Tucker [68] in 1951. Their work grew out of a project on game theory and linear programming initiated after World War II by the Office of Naval Research in the United States. Note that even though nonlinear programming originated in a context of linear programming, the driving forces behind Kuhn and Tucker’s development of nonlinear programming were indeed very different from the stimulus that started the development of linear programming. The duality theorems in linear programming were crucial for Kuhn and Tucker’s development of nonlinear programming. Later, it was found that duality theorems had been proved already, once by Karush [59] and once by John [53]. It is interesting to note that the work by Kuhn and Tucker in 1951 on the necessary and sufficiency conditions for the optimal solution of NLPPs laid the foundations for a great deal of later research in nonlinear programming. They developed the Lagrangian multiplier rule for convex and other NLPPs, which also involved inequality constraints. The Kuhn-Tucker optimality conditions became very useful and important in developing algorithms for solving convex and other NLPPs with differentiable functions. Because of the contributions by Karush, who developed optimality conditions similar to those of Kuhn and Tucker, these conditions are known as the Karush-Kuhn-Tucker conditions.

NLPPs come in many different shapes and forms. Unlike the simplex method for LPPs, no single algorithm can solve all these different types of NLPPs. Instead, algorithms have been developed for various individual classes of NLPPs. These days, many excellent software systems that can solve large classes of NLPPs are available. Some of these are based on the generalized reduced gradient method [70] and GRG2 [99]. LINGO [95] is an interactive interface to GRG2 for solving NLPPs.

1.1.2 Multiobjective optimization problem

Multiobjective optimization [101], also known as vector optimization or multiple-criteria optimization, is the process of systematically and simultaneously optimizing two or more conflicting objectives subject to certain constraints. Multi-objective optimization problems (MOOPs) can be found in various fields such
Introduction

as product and process design, finance, aircraft design, automobile design, or wherever optimal decisions need to be taken in the presence of trade-offs among two or more conflicting objectives. For example, maximizing profit and minimizing the cost of a product, maximizing performance and minimizing fuel consumption of a vehicle, and minimizing weight while maximizing the strength of a particular component. In mathematical terms, MOOP can be written as follows:

$$\text{P(1.4)} \quad \max (\text{or min}) \phi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_p(x))$$

subject to

$$x \in \Omega,$$

where $\phi : \Omega \rightarrow \mathbb{R}^p$ is a given vector function consisting of $p$ objective functions to be maximized or minimized.

In P(1.4), $p$ objective functions need to be optimized simultaneously, but in the case of multiple objectives ($p \geq 2$), there does not necessarily exist a solution that is best with respect to all the objectives because of the conflict among them. A solution may be best for one objective but worst for the remaining objective(s). Therefore, usually, there exists a set of solutions for MOOPs called nondominated solutions or pareto optimal solutions where no improvement is possible in any objective function without deteriorating at least one of the other objective functions. Finding such solutions, and quantifying the trade-offs in satisfying the different objectives, is the goal when setting up and solving a MOOP.

**Definition 1.1. (Pareto optimal solution or efficient solution for a minimization problem)** A point, $x^* \in \Omega$, is called a pareto optimal solution of P(1.4) iff there does not exist another point, $x \in \Omega$, such that $\phi(x) \leq \phi(x^*)$, and $\phi_i(x) < \phi_i(x^*)$ for at least one function.

**Definition 1.2. (Weakly pareto optimal solution or weakly efficient solution for a minimization problem)** A point, $x^* \in \Omega$, is called a weakly pareto optimal solution of P(1.4) iff there does not exist another point, $x \in \Omega$, such that $\phi(x) < \phi(x^*)$. 

A point is weakly pareto optimal if there is no other point that improves all the objective functions simultaneously. In contrast, a point is pareto optimal if there is no other point that improves at least one objective function without detrimentally affecting the remaining objective function(s). Pareto optimal points are weakly pareto optimal, but weakly pareto optimal points are not pareto optimal. The idea of proper pareto optimality and its relevance to certain algorithms are discussed by Geoffrion [35], Yu et al. [114], and Miettinen [83].

Many methods for solving MOOPs are available. These involve converting the MOOP into one or a series of single-objective optimization problems. Each of these problems involves the optimization of a ‘scalarizing’ function, which is a function of original objectives. Traditional methods used to solve MOOPs include no preference methods, posteriori methods, priori methods, and interactive methods.

In no preference methods such as the global criteria method and the multiobjective proximal bundle method, the preferences of the decision maker (DM) are not taken into consideration. The obtained solution is either accepted or rejected by the DM. It is quite unlikely that the solution that meets the preferences of the DM can be found with these methods. The posteriori methods such as the $\varepsilon$-constraint method [18] and the weighting method [35] are usually termed as methods used for generating efficient solutions. In these methods, the set of efficient solutions or a part of it is presented to the DM who selects the most preferred among the alternatives. The inconveniences with these methods are that the generation process is usually difficult and also that it is difficult for the DM to select from a large set of alternatives. In the case of priori methods such as the utility function method, the lexicographic ordering method [46], and the goal programming method [19], the DM must satisfy his/her preferences before the solution process. The difficulty with these methods is that the DM does not necessarily know beforehand what is possible to attain in the problem and how realistic his/her preferences are. The class of interactive methods is the most developed of the classes of methods mentioned above for solving a MOOP. In these methods, once a solution pattern is formed it is repeated several times to meet the preferences of the DM as closely as possible. After a reasonable number of iterations, every interactive method yields a solution that meets the
preferences of the DM in the sense that no better solution exists.

Also, one possible way to solve the problem of choosing alternatives in MOOPs is by using a multiple-criteria decision making (MCDM) process. An MCDM process is a branch in decision engineering that operates in a discrete decision space—a space in which there are a finite number of alternatives. Typically, an MCDM process aids the DM to evaluate and choose an alternative from a set of alternatives with conflicting goals. Some examples of MCDM processes include the analytical hierarchy process (AHP), technique for order preference by similarity to ideal solution (TOPSIS), elimination and choice translating relativity (ELECTRE), and preference ranking organization method for enrichment evaluation (PROMETHEE) [106]. Out of these methods, the AHP is one of the most popular and has been applied widely in solving complex decision making problems [108]. The AHP was developed by Saaty [93] to solve complex decision making problems, which involves ranking and choosing of alternatives. The main advantage of the AHP is its inherent ability to handle intangibles, which are predominant in any decision making process. Also, less cumbersome mathematical calculations and comprehensibility makes the AHP an ideal technique that can be used in the evaluation process. In AHP, the preferences of the DM are elicited in the form of ratios by using a pairwise comparison matrix. The DM compares the elements in the pairwise comparison matrix and assigns numerical values to them. A final aggregation of local weights is performed to rank and choose the best alternative. The following three steps are followed for solving decision problems by using AHP.

**Step 1: Establishing structural hierarchy**

This step allows a complex decision to be structured into a hierarchy descending from an overall objective to various ‘criteria,’ ‘subcriteria,’ and so on until the lowest level. The objective or the overall goal of the decision is represented at the top level of the hierarchy. The criteria and subcriteria contributing to the decision are represented at the intermediate levels. Finally, the decision alternatives or selection choices are laid down at the last level of the hierarchy. According to Saaty [93], a hierarchy can be constructed by creative thinking, recollection, and using people’s perspectives. Note that there is no set of procedures for generating the levels to be included in a hierarchy. The structure
of a hierarchy depends on the nature or type of managerial decisions. Also, the number of levels in a hierarchy depends on the complexity of the problem being analyzed and the degree of detail of the problem that an analyst requires. As such, the hierarchy representation of a decision process may vary from one person to another.

**Step 2: Establishing comparative judgments**

Once the hierarchy has been structured, the next step is to determine the priorities of elements at each level (‘element’ here means every member of the hierarchy). A set of pairwise comparison matrices of all elements in a level of the hierarchy with respect to an element of the immediately higher level is constructed so as to prioritize and convert individual comparative judgments into ratio-scale measurements. The preferences are quantified by using a nine-point scale [93] (see Table 1.1). The pairwise comparisons are given in terms of how much more an element \( A \) is important than an element \( B \).

<table>
<thead>
<tr>
<th>Verbal scale</th>
<th>Numerical values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equally important, likely or preferred</td>
<td>1</td>
</tr>
<tr>
<td>Moderately more important, likely or preferred</td>
<td>3</td>
</tr>
<tr>
<td>Strongly more important, likely or preferred</td>
<td>5</td>
</tr>
<tr>
<td>Very strongly more important, likely or preferred</td>
<td>7</td>
</tr>
<tr>
<td>Extremely more important, likely or preferred</td>
<td>9</td>
</tr>
<tr>
<td>Intermediate values to reflect compromise</td>
<td>2, 4, 6, 8</td>
</tr>
<tr>
<td>Reciprocals for inverse comparison</td>
<td>Reciprocals</td>
</tr>
</tbody>
</table>

**Step 3: Synthesis of priorities and the measurement of consistency**

The elements of each level of the decision hierarchy are rated by using pairwise comparison. After all the elements have been compared pair by pair, a paired comparison matrix is formed. The order of the matrix depends on the number of elements at each level. The number of such matrices at each level depends on the number of elements at the immediate upper level that it links to. After developing all the paired comparison matrices, the eigenvector or the relative weights representing the degree of relative importance among the elements and
the maximum eigenvalue ($\lambda_{max}$) are calculated for each matrix. The $\lambda_{max}$ value is used as a reference index ($RI$) to screen information by calculating the consistency ratio ($CR$) of the estimated vector. This is done to validate whether the paired comparison matrix provides a completely consistent evaluation. The $CR$ is calculated as per the following steps:

1. Calculate the eigenvector or the relative weights and $\lambda_{max}$ for each matrix of order $n$.

2. Compute the consistency index ($CI$) for each matrix of order $n$ as follows:
   \[ CI = \frac{\lambda_{max} - n}{n - 1}. \]

3. The $CR$ is calculated as follows:
   \[ CR = \frac{CI}{RI}, \]

where $RI$ is a known random index that has been obtained from a large number of simulation runs and varies according to the order of the matrix. The acceptable $CR$ value for a matrix at each level is less than or equal to 0.1. If $CI$ is sufficiently small, then pairwise comparisons are probably consistent enough to give useful estimates of the weights. If $CI/RI \leq 0.1$, then the degree of consistency is satisfactory; otherwise, serious inconsistencies may exist and hence AHP may not yield meaningful results. The evaluation process in such cases is then reviewed and improved. The eigenvectors are used to calculate the global weights if there is an acceptable degree of consistency of the selection criteria.

1.2 Fuzzy optimization

Over the last several decades, optimization models have been primarily developed in a deterministic and crisp environment. In such models, the objective(s) and the constraints are formulated in a hard, crisp manner leaving no scope for uncertainty, and the solutions are either feasible or infeasible, either above a certain aspiration level or below. Such models often lead to approximating real-world problems, the solutions of which may not always be acceptable to the end user. This is particularly true if the problem under consideration includes vaguely defined relationships, human evaluation, and imprecision or uncertainties inherent in the parameters. Some researchers pointed out that
uncertainties are of two kinds–ambiguity and vagueness [29, 62]. Ambiguity is associated with those situations in which the choice among two or more alternatives is left unspecified, while vagueness is associated with those domains of interest that cannot be delimited by sharp boundaries. The vagueness in DM’s understanding of the objective(s) and constraints of an optimization problem, as well as the ambiguity inherent in the parameters involved, can be efficiently modeled through fuzzy set theory. In 1965, Zadeh [116] developed the fuzzy set theory as a way to capture uncertainty or vagueness often overlooked in complex systems. Since then it has been used in many application areas such as operations research, management science, control theory, and artificial intelligence/intelligent systems.

Fuzzy set theory is a generalization of the classical set theory. In classical set theory, each element of a universe $X$ either belongs to a set $A$ or not, whereas in fuzzy set theory, an element belongs to a set $A$ with a certain degree of membership. We first discuss the basic preliminaries of fuzzy set theory compiled from Bector and Chandra [6] and then discuss how it can be used to solve fuzzy optimization problems.

**Definition 1.3. (Fuzzy set)** Let $X$ be a universe of discourse. $A$ is a fuzzy subset of $X$ if $\forall x \in X$, there is a number $\mu_A(x) \in [0, 1]$ assigned to represent the membership of $x$ to $A$. $\mu_A(x)$ is called the membership function of $A$.

The fuzzy set $A$ in $X$ is thus uniquely characterized by its membership function $\mu_A(x)$, which associates with each element in $X$, a non-negative real number whose value lies in the interval $[0, 1]$. The value of $\mu_A(x)$ at $x$ represents the ‘grade of membership’ of $x$ in $A$; thus, nearer the value of $\mu_A(x)$ to 1, higher the grade of ‘belongingness’ of $x$ in $A$.

**Definition 1.4. (Support of a fuzzy set)** Let $A$ be a fuzzy set in $X$. Then the support of $A$, denoted by $S(A)$, is the crisp set given by

$$S(A) = \{x \in X : \mu_A(x) > 0\}.$$ 

**Definition 1.5. (Normal fuzzy set)** Let $A$ be a fuzzy set in $X$. The height $h(A)$ of $A$ is defined as

$$h(A) = \sup_{x \in X} \mu_A(x).$$
If \( h(A) = 1 \), then the fuzzy set \( A \) is called a normal fuzzy set; otherwise, it is called subnormal. If \( 0 < h(A) < 1 \), then the subnormal fuzzy set \( A \) can be normalized, that is, it can be made normal by redefining the membership function as \( \mu_A(x)/h(A) \), \( x \in X \).

**Definition 1.6. (\( \alpha \)-cut)** Let \( A \) be a fuzzy set in \( X \) and \( \alpha \in (0, 1] \). The \( \alpha \)-cut of the fuzzy set \( A \) is the crisp set \( A_{\alpha} \) given by

\[
A_{\alpha} = \{ x \in X : \mu_A(x) \geq \alpha \}.
\]

From the definition of \( \alpha \)-cut, it immediately follows that for any fuzzy set \( A \) and pair \( \alpha_1, \alpha_2 \in (0, 1], \alpha_1 \leq \alpha_2 \), we have \( A_{\alpha_2} \subseteq A_{\alpha_1} \). Therefore, all \( \alpha \)-cuts of any fuzzy set form families of crisp sets which can be used to represent a given fuzzy set \( A \) in \( X \).

**Definition 1.7. (Zadeh’s extension principle)** Let \( f : X \to Y \) be a crisp function and \( F(X) \) (respectively \( F(Y) \)) be the set of all fuzzy sets (called fuzzy power set) of \( X \) (respectively \( Y \)). Then, using Zadeh’s extension principle the membership function of fuzzy set \( f(A) \) in \( Y \) (respectively \( f^{-1}(B) \) in \( X \)) in terms of membership function of fuzzy set \( A \) in \( X \) (respectively \( B \) in \( Y \)) are given by

\[
(i) \quad \mu_{f(A)}(y) = \sup_{x \in X, \, f(x) = y} (\mu_A(x)), \, \forall A \in F(X), \text{ and}
\]

\[
(ii) \quad \mu_{f^{-1}(B)}(x) = \mu_B(f(x)), \, \forall B \in F(Y).
\]

Suppose the function \( f \) maps \( n \)-tuple in \( X \) to a point in \( Y \), that is, \( X = X_1 \times X_2 \times \ldots \times X_n \) and \( f : X \to Y \) given by \( y = f(x_1, x_2, \ldots, x_n) \). Let \( A_1, A_2, \ldots, A_n \) be \( n \) fuzzy sets in \( X_1, X_2, \ldots, X_n \), respectively. The extension principle of Zadeh allows to extend the crisp function \( y = f(x_1, x_2, \ldots, x_n) \) to act on \( n \) fuzzy subsets of \( X \), namely, \( A_1, A_2, \ldots, A_n \) such that \( B = f(A_1, A_2, \ldots, A_n) \).

Thus, the fuzzy set \( B \) is defined by

\[
B = \{ (y, \mu_B(y)) : y = f(x_1, x_2, \ldots, x_n), \, (x_1, x_2, \ldots, x_n) \in X_1 \times X_2 \times \ldots \times X_n \}
\]

and \( \mu_B(y) = \sup_{x \in X, \, f(x) = y} \min(\mu_{A_1}(x_1), \, \mu_{A_2}(x_2), \ldots, \mu_{A_n}(x_n)) \).
Definition 1.8. (Fuzzy number) A fuzzy set $A$ in $\mathbb{R}$ is called a fuzzy number if it satisfies the following conditions

(i) $A$ is normal,

(ii) $A_\alpha$ is a closed interval for every $\alpha \in (0, 1]$,

(iii) the support of $A$ is bounded.

The following result gives a complete characterization of a fuzzy number.

Theorem 1.4. Let $A$ be a fuzzy set in $\mathbb{R}$. Then $A$ is a fuzzy number iff there exists a closed interval (which may be singleton) $[a, b] \neq \emptyset$ such that

$$
\mu_A(x) = \begin{cases} 
1, & x \in [a, b], \\
l(x), & x \in (-\infty, a), \\
r(x), & x \in (b, \infty),
\end{cases}
$$

where $l : (-\infty, a) \to [0, 1]$ is increasing continuous from the right and $l(x) = 0$ for $x \in (-\infty, w_1)$, $w_1 < a$; and $r : (b, \infty) \to [0, 1]$ is decreasing continuous from the left and $r(x) = 0$ for $x \in (w_2, \infty)$, $w_2 > b$.

It is worth mentioning that in case the membership function of the fuzzy set $A$ in $\mathbb{R}$ takes the form $\mu_A(x) = 1$ for $x = a$ and $\mu_A(x) = 0$ for $x \neq a$, it becomes the characteristic function of the singleton set $\{a\}$ and therefore represents the real number $a$. A real interval $[a, b]$ can also be identified similarly by its characteristic function. In most of the practical applications, the functions $l(x)$ and $r(x)$ are continuous which give the continuity of the membership function.

Definition 1.9. Let $A$ be a fuzzy number. Then $A_L$ and $A_U$ are defined as

$$
A_L = \inf_{\mu_A(x) \geq \alpha} (x) \quad \text{and} \quad A_U = \sup_{\mu_A(x) \geq \alpha} (x).
$$

Definition 1.10. (Triangular fuzzy number) A fuzzy number $A$ is called a triangular fuzzy number (TFN) if its membership function $\mu_A(x)$ is given by

$$
\mu_A(x) = \begin{cases} 
0, & \text{if } x < a_l, \ x > a_u, \\
\frac{x-a_l}{a_u-a_l}, & \text{if } a_l \leq x \leq a, \\
\frac{a_u-x}{a_u-a}, & \text{if } a < x \leq a_u.
\end{cases}
$$
The TFN $A$ is denoted by the triplet $A = (a_l, a, a_u)$ and has the shape of a triangle as shown in Figure 1.1 (replicated from [6]).

![Figure 1.1: A triangular fuzzy number](image)

The $\alpha$-cut of the TFN $A = (a_l, a, a_u)$ is the closed interval

$$A_\alpha = [A^L_\alpha, A^R_\alpha] = [(a - a_l)\alpha + a_l, (a - a_u)\alpha + a_u], \ \alpha \in (0, 1].$$

**Definition 1.11. (Trapezoidal fuzzy number)** A fuzzy number $A$ is called a trapezoidal fuzzy number (TrFN) if its membership function $\mu_A(x)$ is given by

$$\mu_A(x) = \begin{cases} 
0, & \text{if } x < a_l, \ x > a_u, \\
\frac{x - a_l}{a - a_l}, & \text{if } a_l \leq x < a, \\
1, & \text{if } a \leq x \leq \overline{a}, \\
\frac{a_u - x}{a_u - \overline{a}}, & \text{if } \overline{a} < x \leq a_u.
\end{cases}$$

The TrFN $A$ is denoted by the quadruplet $A = (a_l, a, \overline{a}, a_u)$ and has the shape of a trapezoid as shown in Figure 1.2 (replicated from [6]).

![Figure 1.2: A trapezoidal fuzzy number](image)
The $\alpha$-cut of the TrFN $A = (a_l, \underline{a}, \overline{a}, a_u)$ is the closed interval

$$A_\alpha = [A^L_\alpha, A^R_\alpha] = [(a - a_l)\alpha + a_l, (\overline{a} - a_u)\alpha + a_u], \quad \alpha \in (0, 1].$$

**Ranking fuzzy numbers** plays an important role in practical use such as in approximate reasoning, decision making, forecasting, control, and other usage. In fuzzy decision analysis, fuzzy numbers are frequently used to describe the performance of alternatives in modeling a real-world problem. The DM assesses the alternatives with fuzzy numbers, and the selection of alternatives will eventually lead to the ranking of the corresponding fuzzy numbers. Thus, specific ranking of fuzzy numbers is an important procedure for decision making in a fuzzy environment and generally has become one of the main tools in fuzzy decision making. Because fuzzy numbers are represented by possibility distributions, they can overlap with each other and it is difficult to determine clearly whether one fuzzy number is larger or smaller than the other [44]. The method of ranking fuzzy numbers was first proposed by Jain [50]. Since then a large variety of methods ranging from the trivial to the complex have been developed, including one fuzzy number attribute to many fuzzy number attributes. In a study conducted by Chen et al. [20], the ranking methods were classified into the following four major classes: preference relation, fuzzy mean and spread, fuzzy scoring, and linguistic expression.

Fuzzy set theory appears to be an ideal approach to deal with decision problems that are formulated as optimization problems under uncertainty. **Fuzzy optimization problems** are designated for such a purpose. On the basis of the kind of uncertainty, fuzzy optimization problems are classified as vagueness or ambiguity problems [48]. In vagueness problems, fuzzy goals and constraints are considered, and usually a satisfaction degree is obtained both for the goals and for the constraints [7, 121]. In ambiguity problems, objective function coefficients and/or technological coefficients are fuzzy numbers. Some examples of the latter case can be found in Inuiguchi et al. [47], Li and Yu [74], and Malecki et al. [80].

Let us consider the crisp optimization problem defined as follows:
\textbf{P(1.5)} \quad \max c^T x \\
subject to \\
Ax \leq b, \\
x \geq 0,

where \( x \in \mathbb{R}^n, \ c \in \mathbb{R}^n, \ b \in \mathbb{R}^m, \) and \( A \in \mathbb{R}^{m \times n}. \)

Decision making under uncertainty brings modifications in the above-defined problem, thus resulting in a fuzzy optimization problem. These modifications can be in various forms, such as the following:

(a) Instead of maximizing the objective function, one may desire to achieve some aspiration level that might be vague.

(b) Constraints might be vague; that is, the ‘\( \leq \)’ sign might not be meant in the strict mathematical sense.

(c) The elements of the vectors \( c \) or \( b \) or of the matrix \( A \) may not be crisp but rather may be fuzzy numbers.

Consequently, fuzzy optimization problems cannot be defined uniquely. They can be broadly classified as follows:

1. Fuzzy optimization problem with fuzzy inequalities and crisp objective function defined as:

\textbf{P(1.6)} \quad \max c^T x \\
subject to \\
Ax \preceq b, \\
x \geq 0.

2. Fuzzy optimization problem with crisp inequalities and fuzzy objective function defined as:
3. Fuzzy optimization problem with fuzzy inequalities and fuzzy objective function defined as:

\[ \textbf{P(1.7)} \quad \max \ c^T x \]
subject to
\[ Ax \leq b, \]
\[ x \geq 0. \]

4. Fuzzy optimization problem with fuzzy parameters defined as:

\[ \textbf{P(1.8)} \quad \max \ c^T x \]
subject to
\[ \tilde{A}x \preceq \tilde{b}, \]
\[ x \geq 0. \]

Furthermore, fuzzy optimization problems can be classified as symmetric or nonsymmetric problems. On the one hand, in symmetric problems, which are based on the definition of fuzzy decision proposed by Bellman and Zadeh [7], the basic feature is the symmetry between the objective(s) and the constraints. Because of this approach, the decision set becomes a fuzzy set resulting from the intersection of the fuzzy sets corresponding to the objective(s) and the constraints. On the other hand, in nonsymmetric problems, there is a distinction between the objectives and the constraints.

Problem formulations based on fuzzy sets can have greater expressive power than do their counterparts based on crisp sets, but the applicability of the fuzzy approach depends on the ability to construct membership functions [63] that
appropriately represent various concepts in different contexts. Various types of membership functions have been used in fuzzy optimization problems such as linear membership function [34, 119, 120], tangent-type membership function [71], interval linear membership function [40], exponential membership function [14], inverse tangent membership function [94], piecewise linear membership function [42, 43], and logistic-type membership function, that is, a nonlinear S-shape membership function [111]. Tangent-type membership function, exponential membership function, and inverse tangent membership function are nonlinear functions; a fuzzy optimization problem defined with a nonlinear membership function results in an NLPP. Usually, a linear membership function is used to avoid nonlinearity.

Suppose a decision making problem is characterized by fuzzy goals \( \tilde{G}_i \) \((i = 1, 2, \ldots, n)\) along with a set of fuzzy constraints \( \tilde{C}_j \) \((j = 1, 2, \ldots, m)\) in a space of alternatives \( X \). For solving such problems, Bellman and Zadeh proposed that a fuzzy decision \( \tilde{D} \) is determined by the aggregation of the fuzzy sets \( \tilde{G}_i \) \((i = 1, 2, \ldots, n)\) and \( \tilde{C}_j \) \((j = 1, 2, \ldots, m)\). Mathematically, \( \tilde{D} = (\tilde{G}_1 \cap \tilde{G}_2 \cap \ldots \tilde{G}_n) \cap (\tilde{C}_1 \cap \tilde{C}_2 \cap \ldots \tilde{C}_m) \), that is, \( \mu_{\tilde{D}}(x) : X \to [0, 1] \) given by \( \mu_{\tilde{D}}(x) = \min_{i,j}(\mu_{\tilde{G}_i}(x), \mu_{\tilde{C}_j}(x)) \). Once we obtain the fuzzy decision \( \tilde{D} \), we can define \( x^* \in X \) to be an optimal decision if \( \mu_{\tilde{D}}(x^*) = \max_{X} \mu_{\tilde{D}}(x) \). Alternatively, choose an \( \alpha (0 < \alpha < 1) \) and determine all points \( x^* \in X \) for which \( \mu_{\tilde{D}}(x^*) \geq \alpha \).

In Bellman and Zadeh's approach, the fuzzy optimization problem is converted into a crisp optimization problem by using a max-min operator. If the resulting crisp optimization problem has only one optimal solution, then this solution has to be a fuzzy efficient compromise solution for the given fuzzy optimization problem. However, if the resulting crisp optimization problem has multiple optimal solutions, then the solution obtained by using the max-min approach may not be efficient but at least one of the multiple optimal solutions is certainly a fuzzy efficient compromise solution. In that case, we can use a two-phase approach to obtain a fuzzy efficient compromise solution. In the two-phase approach, the crisp optimization problem resulting from the max-min approach is solved in phase I, while in phase II a solution is desired that is at least ‘better’ than the solution obtained by using the max-min approach and every membership function is at least as large as the one provided by the max-min approach.
1.3 Component-based software development

In today’s era, modern software systems are becoming more and more complex and wider in scale, resulting in high development cost, low productivity, unmanageable software quality, and high risk to move to a new technology. Also, some systems include such complex components that developing them from scratch would be impossible if a profit is desired. Consequently, there is a growing demand for a new, efficient, and cost-effective software development paradigm. One of the most promising solutions today is the component-based software development (CBSD) approach. This approach is based on the idea that while developing software systems using components, the resulting integrated system must satisfy certain requirements (functional and nonfunctional). However, the focus of development moves to issues such as evaluation, selection, integration, and evolution of components in the system. Underestimating the technical risks associated with evaluation, selection, and integration of software components can result in long schedule delays and high development/maintenance cost. Thus, depending on system requirements, components may be bought as commercial-off-the-shelf (COTS) components. Most of today’s software systems include one or more COTS components (products). If some requirement(s) cannot be satisfied with COTS components, then the component(s) corresponding to the given system requirement(s) may be developed in-house.

CBSD involves composition and reconciliation, whereas custom software development is an act of creation. Custom software development starts with the system requirements and creates a system that meets them; the engineers are producers. However, CBSD starts with a general set of requirements and then explores the marketplace’s offerings to see how closely they match the needs; the engineers are consumers who integrate the components they buy into a system that meets the need. The nature, timing, and order of activities performed and the processes used differ accordingly. A fundamental change is required in the approach used for system development in component-based software systems, as shown in Figure 1.3. Figure 1.3 (left) shows a traditional custom-development approach, in which requirements are identified first, an architecture is defined next, and (custom) implementation is undertaken finally. However, if this approach is applied to component-based software systems, it is unlikely that the
marketplace will yield any software system that fit the \textit{a priori} requirements and architecture. Instead, with component-based software systems, it is necessary to consider requirements, architecture, and the marketplace simultaneously as pictured on the right in Figure 1.3. Any of these three considerations may have an impact on the other two, and so none can proceed without knowledge and accommodation of the other two. Furthermore, the activities that are performed for component-based software systems are cyclic in nature and these trade-offs will be repeated frequently throughout the lifetime of the system. Thus, when the process changes, people must change as well. This means that CBSD is not just an engineering or technical change; it is a business, organizational, and cultural change too.

![Diagram of Traditional versus COTS-based approach](image)

\textbf{Figure 1.3:} Traditional versus COTS-based approach

To ensure that a component-based software system can run properly and effectively, the system architecture is the most important factor [37]. The system architecture of component-based software systems should be a layered and modular architecture. This architecture can be seen in Figure 1.4. The top layer is the application(s) supporting a business. The second layer consists of programs engaged in only a specific business or application domain that performs one or more functions as is the requirement. The third layer includes cross-business middleware components called modules, which are executed sequentially. Some of these modules may be common to different programs. Finally, the lowest layer contains a set of alternative components available for each module that interface with the underlying operating systems and hardware.
In the last several years, the world of software system development has been revolutionized by COTS components. Different researchers have attempted to define what constitutes a COTS component [65, 89, 96]. CeBASE [17] defines a COTS component as follows: A software component developed by a third party (who controls its ongoing support and evolution), bought, licensed or acquired for the purposes of integration into a larger system as an integral part, that is, that will be delivered as part of the system to the customer of that system, which might or might not allow modification at the source code level, but may include mechanisms for customization and is bought and used by a significant number of system developers. This is shown in Figure 1.5, where COTS components can be checked out from a component repository and assembled into a target software system.

**Figure 1.4**: System architecture of component-based software systems

**Figure 1.5**: COTS selection for development of component-based software systems
The Institute of Electrical and Electronics Engineers [45] defines the characteristics of a COTS component as follows: COTS component is stable and normally well defined in terms of documentation, known capabilities and limitations. It usually comes with ‘how to operate’ documentation. COTS component is defined by a market-driven need. It is commercially available and its fitness for use has been demonstrated by a broad spectrum of commercial users. Also, the COTS components supplier does not advertise any willingness to modify the software for a specific customer.

From the above definitions, it is inferred that COTS components are produced and/or distributed by vendors and integrators compose a set of components into a COTS-based software system that is eventually deployed to the customer. Thus, there are three main roles in the COTS/component-based development process [105] (see Figure 1.6).

1. **Vendor**: The organization that produces, sells, or distributes the COTS component. It is the source for both the component itself and its documentation. Often, it provides some type of support services, which may be free or sold separately.
2. **Integrator**: The organization that builds the system by using COTS components, possibly together with internally developed pieces of software (glueware) and internally developed components.

3. **Customer**: The organization that acquires, pays for, and dictates the requirements of the COTS/component-based software system. The customer may or may not coincide with the user.

Typical examples of COTS components [15] include the following:

1. Stand-alone packages such as word processors (e.g., Microsoft Word) and spreadsheets.
2. Digital imaging packages such as Adobe Photoshop and COREL PhotoPaint.
3. Libraries requiring linkage into application code, for example, graphics engines and Windows DLLs.
4. Development environments with runtime modules, for example, Visual Basic and Sybase.
5. Vendor-supplied device drivers such as printers, displays, and multimedia.
6. Information retrieval applications such as Internet browsers (e.g., Internet Explorer and Mozilla Firefox), e-mail packages, and data mining tools.
7. Operating system utilities such as file operations and memory management.
8. Antivirus systems and so on.

These components are particularly popular in application areas, such as payroll, banking, insurance, accounting, networking, inventory control, and systems software [81].

On the one hand, the use of COTS components, instead of developing the components in-house, has several potential advantages such as lower purchasing cost due to the mass production, lower development effort because COTS components are ready-made, high maturity because COTS components go through multiple releases starting from beta versions to final versions, and rich functionality because COTS components cover a wide spectrum of users with different
needs [110]. On the other hand, several challenges are encountered when using COTS components, including the lack of control over COTS component development and evolution [11] and COTS components’ inability to meet specific requirements in different projects [3]. To cope with such challenges, suitable methods and techniques should be used to evaluate and select the most appropriate COTS components from among the existing COTS components. Developers must assess a number of properties of the wanted COTS component to integrate it properly with a system under development. Inappropriate COTS selection can result in serious consequences leading to system failure, for example, the London Ambulance Service fiasco in 1992, in which the system descended into chaos and reverted to manual operation partly because of inappropriate COTS selection [79].

*COTS decision making* involves selecting an appropriate COTS component, which is often a nontrivial task and can be described as an MCDM problem [88]. To realize the benefits of COTS components, it is important that the ‘right’ components are selected for a software system. The selection of best-fit COTS components has now become the key to the quality and cost of modular software systems based on COTS. With an abundance of COTS components to choose from, the COTS selection problem involves selecting COTS components that best meet the functional requirements, for example, cost and reliability of a software system. Although, several *COTS selection* methods [25, 36, 64, 67, 72, 90] have been proposed in the literature, there is no single method that is accepted as a standard COTS selection method. A detailed list of COTS selection methods has been given in Mohamed et al. [84].

Although there is no commonly accepted method for COTS selection [91], all methods share some key steps that can be iterative and overlapping. Still, there are many factors and issues that should be taken into consideration during selection of COTS components. Kontio et al. [64] proposed the following phases for COTS selection process (see Figure 1.7).
1. **Criteria definition**: In the first phase, a hierarchical set of criteria is derived from reuse goals that are affected by various factors such as an organization’s infrastructure, application architecture and design, application requirements, project objectives and constraints, or the availability of software libraries. Establishing the criteria for COTS selection is a critical task in CBSD.

2. **Searching**: This phase aims at identifying COTS components. In this phase, the number of possible alternatives may grow quite rapidly.

3. **Screening**: The most potential candidates will need to be sorted out in this phase to pick the ones that can be evaluated in more detail with the resources available.

4. **Evaluation**: The evaluation phase strives for consolidating and evaluating the preselected alternatives with respect to the defined criteria and documenting these results.

5. **Analysis**: In the analysis phase, the DM decides on the ‘best-fit’ COTS component for the application under consideration. The analysis phase is separated out to emphasize the importance of interpreting evaluation data. Sometimes, it may be possible to make straightforward conclusions if one of the alternatives is clearly superior to others; however, in most cases, it is necessary to use systematic MCDM techniques to arrive at a decision. On the basis of the decisions made, typically one of the alternatives is selected and
deployed. Finally, to improve the selection process and to provide feedback on potential further reuse of the component, it is necessary to assess the success of the reuse component used in a project.

**COTS evaluation** is the core of the COTS selection process that determines the fitness of COTS components. COTS evaluation provides the necessary information for DMs to make an appropriate decision to select a COTS component from a set of competing alternatives. Kontio et al. [66] defined the evaluation criteria in a hierarchical manner, where a set of abstract goals is gradually refined on the basis of such factors as the software application requirements, the application architecture, and the existing components’ capabilities. Franch and Carvallo [32] and Carvallo et al. [16] proposed a six-step method to build a structured quality model for the purpose of COTS evaluation. They relied on the ISO/IEC 9126 quality model [49] and further explained activities to define a set of metrics that can be used during the evaluation process. Jadhav and Sonar [52] provided the generic list of COTS evaluation criteria that can be used for the evaluation of COTS components. The meaning of each evaluation criterion and its associated measures that are essential for the assessment of COTS components are defined as follows:

**Functional criteria**
Criteria related to the functional capabilities of COTS components are different for different components. Therefore, functional criteria cannot be generalized; hence, these criteria can be considered as various objectives and constraints to be achieved in the decision making process.

**Cost and benefits criteria**
The cost and benefits criteria are used to assess cost- and benefit-related characteristics of COTS components. The purchasing cost of COTS components contains licensing arrangement cost, product and technology cost, and consulting cost, which involves adapting and integration cost, supporting cost, training cost, and maintenance (upgrades) cost. On many occasions, the maintenance cost is not included in the purchasing price. Thus, the maintenance cost during the software implementation should be added to the component cost. If the vendor does not provide free training services for the user group, the training cost would be an important component of the total cost.
Quality criteria

The quality criteria derived from the ISO/IEC 9126 quality model can be used to assess the quality of COTS components. Currently, the ISO/IEC 9126 quality model is widely used by researchers and practitioners to evaluate software quality. The ISO/IEC 9126 quality model is defined by means of general criteria of software, which are further refined into subcriteria, which, in turn, are decomposed into attributes, yielding a multilevel hierarchy. At the bottom of the hierarchy appear the measurable alternative COTS components, whose scores are computed by using some metric, for example, pairwise comparisons that are scale independent.

Standards for software quality models define software quality characteristics as composed of six external attributes of interest, namely, functionality, reliability, usability, maintainability, portability and efficiency. **Functionality** of the COTS component is nothing but the ability of the component to perform according to the specific needs of the organization. **Reliability** is ability of the COTS component to run consistently without crashing under specific conditions. It is used to measure the reliability level to which the system satisfies the functional requirements of the organization. **Usability** is understandability of the COTS component as well as easiness to learn and operate it under certain specific conditions. **Maintainability** is the ability of the COTS component to be modified. Modifications can include corrections, improvements, or adaptation of the software to adjust to changes in the environment in terms of functional requirements and specifications. **Portability** is the ability of the COTS component to be transferred from one environment to another. **Efficiency** is ability of the COTS component to provide appropriate performance relative to the amount of resources used under certain conditions.

Vendor criteria

Vendor support after purchase is critical for the successful installation and maintenance of the software system. Thus, in addition to system cost and quality criteria, factors such as vendor’s condition and vendor’s ability may be considered. These factors can be gathered on the basis of vendor’s reputation. Vendor’s ability criteria implies vendor’s technology level, implementation and service ability, consulting service, and training support. Vendor’s condition
considers vendor’s financial condition, certifications, and credentials.

Various optimization models for COTS selection [9, 23, 51, 86, 118] exist in the literature that achieve different attributes of quality along with the objective of minimizing the cost or keeping the cost to a specified budgetary level. Most of the researchers formulated optimization models of COTS selection for the development of modular software systems under a single application development task. In what follows next, we discuss some of the optimization models for COTS selection.

Jung and Choi [54] presented two optimization models for selecting the best-fit COTS components from among available alternatives for each module in the development of modular software systems. They considered the overall quality of different possible alternative COTS components of each module and used the objective of maximizing the weighted overall quality as a single entity. The weight assignment to each module utilizes the AHP [92], which was earlier used by Zahedi and Ashrafi [118] to evaluate the reliability of modular software systems.

The first model of this study is formulated as follows:

$$
\text{P}(1.10) \quad \max Q(x) = \sum_{i=1}^{m} w_i \left( \sum_{j=1}^{n_i} q_{ij} x_{ij} \right)
$$

subject to

$$
\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1, 2, \ldots, m,
$$

$$
\sum_{i=1}^{m} \sum_{j=1}^{n_i} c_{ij} x_{ij} \leq B,
$$

$$
x_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n_i,
$$

where $m$ is the number of modules designed for the software system, $n_i$ is the number of alternative COTS components for the $i$-th module, $w_i$ is the weight assigned to the $i$-th module, $q_{ij}$ is the quality level of the $j$-th COTS component for the $i$-th module estimated by using any of the quality models available such as ISO/IEC 9126 or the models proposed by Evans and Marciniak [31], $c_{ij}$ is the cost of the $j$-th COTS component for the $i$-th module provided by the vendor,
B is the total available budget, and $x_{ij}$ is a binary decision variable; $x_{ij} = 1$ if the $j$-th COTS component is selected for the $i$-th module; otherwise, $x_{ij} = 0$.

P(1.10) maximizes the weighted quality with budget consideration, while the second model P(1.11), which is an extension of the first model, includes an additional ‘either-or’ constraint that may reflect the real procurement environment. In P(1.10), it is assumed that all alternative COTS components for one module are compatible with the alternative COTS components of other modules. However, sometimes, COTS components are incompatible due to problems such as implementation technology, interfaces, and licensing. This condition, which is called a contingent decision constraint [41], may be represented by $x_{rs} \leq x_{ut_k}$, which means that if COTS component $s$ for module $r$ is selected, then it should come with COTS component $t_k$ for module $u$.

Using this additional constraint, the extended model P(1.11) is defined as follows:

$$
P(1.11) \quad \max Q(x) = \sum_{i=1}^{m} w_i \left( \sum_{j=1}^{n_i} q_{ij} x_{ij} \right)
$$

subject to

$$
\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1, 2, \ldots, m,
$$

$$
\sum_{i=1}^{m} \sum_{j=1}^{n_i} c_{ij} x_{ij} \leq B,
$$

$$
x_{rs} - x_{ut_k} \leq My_k, \quad k = 1, 2, \ldots, z,
$$

$$
\sum_{k=1}^{z} y_k = z - 1,
$$

$$
y_k \in \{0, 1\}, \quad k = 1, 2, \ldots, z,
$$

$$
x_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n_i,
$$

where $z$ is the total number of contingent decision constraints and $y_k$ is a binary variable; $y_k = 0$ if the $k$-th constraint is active; otherwise, $y_k = 1$ if the $k$-th constraint is inactive.

In the above optimization models, the overall quality is computed by combining individual measures of quality such as correctness, reliability, efficiency,
integrity, usability, maintainability, testability, flexibility, and portability. The ISO/IEC 9126 quality model was used to assign weights to quality characteristics to evaluate the overall quality of COTS components. In practice, conflict occurs among the individual measures of quality; hence, compromises may have to be reached. Also, the above method may fail to fulfill the aspiration level of the DM in regard to some of the quality criteria. For example, a DM may be particular that the software he/she develops through the combination of COTS components should attain a minimum reliability level of 0.95. Because there is no separate consideration for reliability criteria in the models, this may not be attained. Also, many of the quality characteristics mentioned in the standard models may not be relevant in a specific COTS project and the DM may desire to have his/her own set of quality attributes.

Zachariah and Rattihalli [115] along with reliability considered other desirable properties (attributes of quality) of software for the purpose of optimization that were set as goals to achieve. These include reliability, execution time, and size (number of lines of code) of the software. Because these goals are not consistent among themselves, they are prioritized and a solution is obtained by using the goal-programming approach. They used redundant modules to create fault-tolerant software systems. Because COTS components are independently developed, may be with different algorithms, failure of one alternative need not imply the failure of the other; hence, they can serve the purpose of redundancy. The addition of redundant modules increases reliability, but at the same time it defeats other goals such as attaining lower cost and smaller size. This study proposed the following model for COTS selection:

\[
P(1.12) \quad \min Z = P_1 \left( \sum_{i=1}^{m} d_i^- \right) + P_2 (d_r^-) + P_3 (d_c^+) + P_4 (d_i^+) + P_5 (d_t^-)
\]

subject to

\[
\sum_{j=1}^{n_i} x_{ij} + d_i^- - d_i^+ = 1, \quad i = 1, 2, \ldots, m,
\]

\[
\sum_{f=1}^{F} p_f \prod_{i \in S_f} \left( 1 - \prod_{j=1}^{n_i} (1 - r_{ij})^{x_{ij}} \right) + d_r^- - d_r^+ = R_d,
\]

\[
\sum_{i=1}^{m} \sum_{j=1}^{n_i} c_{ij} x_{ij} + d_c^- - d_c^+ = C_d,
\]
In the above model, $F$ denotes the total number of functions in the software system; $p_f$ denotes the probability of use of the $f$-th function; $S_f$ denotes the set of modules corresponding to the $f$-th function; $r_{ij}$, $c_{ij}$, $l_{ij}$, and $t_{ij}$ denote the estimated reliability, cost, size (number of lines of code), and expected execution time of the $j$-th COTS component for the $i$-th module, respectively; $R_d$, $C_d$, $L_d$, and $T_d$ denotes the desired reliability, cost, size, and execution time of the software system, respectively; and $P_q$ denotes the priority of the $q$-th goal; $q = 1, 2, \ldots, 5$ ($P_1 \gg P_2 \gg \ldots \gg P_5$). Also, $d_i^-$, $d_i^+$, $d_i^-$, $d_i^+$, $d_i^-$, $d_i^+$, $d_i^-$, $d_i^+$ denote the negative and the positive deviational variables in the goals of selecting exactly one or at least one COTS component for the $i$-th module, attaining the desired software reliability, budgetary requirement, size requirement, and time requirement, respectively.

The above stated optimization models are based on the assumption that in the development of modular software systems, the DM has complete information. Note that because software development is not an exact science, there are often plenty of indefinite and uncertain factors in the estimation of model parameters. Hence, the various model parameters are often imprecise or the process of estimation of these input parameters is subjected to uncertainty in a nonstochastic sense. Optimization models for the development of modular software systems have benefited greatly from the fuzzy set theory in terms of integrating quantitative and qualitative information, subjective preferences of the DM, and knowledge of software experts. Thus, crisp optimization models for COTS selection may be extended to imprecise/uncertain/fuzzy COTS selection models to provide satisfactory solutions to the DM in different scenarios of input data uncertainty. Shen et al. [97] presented a fuzzy optimization model for COTS selection. The objective function of the model is to maximize quality.
within a fuzzy budgetary constraint and the limitation of the incompatibility among COTS components, which may reflect the real procurement environment of COTS components. The fuzzy optimization model for COTS selection is formulated as follows:

\[
P(1.13) \quad \max Q(x) = \sum_{i=1}^{m} w_i \left( \sum_{j=1}^{n_i} q_{ij} x_{ij} \right)
\]

subject to

\[
\sum_{i=1}^{m} \sum_{j=1}^{n_i} c_{ij} x_{ij} \lesssim B,
\]

\[
\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1, 2, \ldots, m,
\]

\[
x_{rs} - x_{ut} \leq M y_k, \quad k = 1, 2, \ldots, z,
\]

\[
\sum_{k=1}^{z} y_k = z - 1,
\]

\[
y_k \in \{0, 1\}, \quad k = 1, 2, \ldots, z,
\]

\[
x_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n_i,
\]

where the symbol ‘\(\lesssim\)’ denotes ‘approximately less than or equal.’ The budgetary constraint is considered as a fuzzy constraint. Problem P(1.13) has the asymmetrical form of fuzzy linear programming, which can be solved by using fuzzy integer 0-1 programming. The authors adopted the parameter programming method proposed by Verdegay [109] to solve problem P(1.13).

Furthermore, in CBSD, if some requirement(s) cannot be satisfied with COTS components, then the software components corresponding to the system requirements may also be developed in-house. Such decision making situations lead to a decision that is popularly known as ‘build-or-buy’ decision. Cortellessa et al. [27] developed an optimization model that supports build-or-buy decisions in selecting software components. They used variables representing the amount of unit testing to be performed on each in-house developed component. Because of the presence of these variables in the cost objective and reliability constraint, the model determines not only the best assembly of components to be bought or built but also the best amount of testing to be performed on each
in-house developed component. For the purpose, a nonlinear 0-1 optimization model was proposed in which the objective function represents the cost of the whole software system to be minimized and each major constraint represents the value of a property of the whole software system to be kept under/over a certain threshold to meet some nonfunctional requirements. The optimization model for COTS selection is formulated as follows:

\[ \text{min } C(x) = \sum_{i=1}^{m} \left( c_i(t_i + \tau_i N_i^{tot})y_i + \sum_{j=1}^{n_i} c_{ij} x_{ij} \right) \]

subject to

\[ \max_{i=1,2,\ldots,m} \left( (t_i + \tau_i N_i^{tot})y_i + \sum_{j=1}^{n_i} d_{ij} x_{ij} \right) \leq T, \]

\[ \prod_{i=1}^{m} \exp \left( - \left( (1 - \rho_i)s_i y_i + \sum_{j=1}^{n_i} s_{ij} x_{ij} \mu_{ij} \right) \right) \geq R, \]

\[ y_i + \sum_{j=1}^{n_i} x_{ij} = 1, \ i = 1,2,\ldots,m, \]

\[ x_{ij}, \ y_i \in \{0,1\}, \ i = 1,2,\ldots,m, \ j = 1,2,\ldots,n_i , \]

where \( m \) denotes the number of modules and \( n_i \) denotes the number of alternative COTS components that are available for the \( i \)-th module. Also, for COTS components, \( d_{ij} \) is the delivery time provided by the vendor; \( s_i \) is the average number of invocations; \( \mu_{ij} \) is the probability of failure on demand, and for in-house developed components, \( c_i \) is the unitary cost, that is, per-day cost of a software developer; \( t_i \) is the estimated development time; \( \tau_i \) is the average time required to perform a test case; \( \pi_i \) is the testability, which is the probability that a single execution of a software fails on a test case chosen from a certain input distribution; and \( y_i \) is a decision variable where \( y_i = 1 \), if the \( i \)-th module is developed in-house; otherwise, \( y_i = 0 \); \( N_i^{tot} \) denotes the total number of tests performed on the in-house developed component for the \( i \)-th module; \( R \) denotes a minimum threshold value of the reliability of the whole software system; and \( \rho_i = \frac{1 - \pi_i}{(1 - \pi_i) + \pi_i (1 - \pi_i) N_{suc}^i} \) is the probability that the in-house developed instance is failure free during a single run given that \( N_{suc}^i \) test cases have been successfully performed where \( N_{suc}^i = (1 - \pi_i)N_i^{tot} \).

In addition, for a modular software system, COTS components should be selected such that the interactions of COTS components within a software module
Introduction

are maximized and interactions of COTS components among software modules are minimized. Coupling is the measure of interactions among software modules, and cohesion is the measure of interactions among software components that are within a software module; hence, for a good software system, modules with high cohesion and low coupling are desired. Thus, in CBSD, a quantitative way of minimizing coupling and maximizing the cohesion of modules must be addressed properly. Kwong et al. [69] proposed a methodology to perform the optimal selection of COTS components for the development of a modular software system based on the criteria of minimal coupling and maximal cohesion of modules achieved using intra-modular coupling density ($ICD$) as well as maximizing the functional performance of the software system yielded by selected COTS components. In their model, $ICD$ is expressed in terms of the ratio of cohesion to all interactions (including cohesion and coupling) within a module.

In practice, not all COTS components can be integrated in an application. Therefore, the compatibility among components has to be considered in the process of COTS selection. Although several previous researchers dealt with the component selection problem of CBSD, relatively few formal methods and techniques were introduced to consider compatibility in the process of component selection and assembly. Jung and Choi made an outstanding contribution to fill this gap. They used either-or constraints to express a compatible relationship between software components. But, in general, there are a large number of compatible relationships among components; thus, the number of either-or constraints will explode rapidly. Furthermore, a software developer may undertake multiple application development tasks of enterprise applications concurrently where the component reusability among applications can also be considered. Tang et al. [103] introduced the concept of reusability and a new formulation of compatibility matrix. They proposed an optimization model for COTS selection considering reusability and compatibility simultaneously. The model is useful for COTS selection when multiple applications are undertaken concurrently.

1.4 Structure of the thesis

Here, we present an overview of the contents of the thesis as organized in Chapters 2 to 5.
In Chapter 2, we consider a modular software system consisting of several programs. A specific function of each program can call upon a series of modules. Suppose the software system consists of $m$ modules that are executed sequentially and the $i$-th module contains $n_i$ alternative COTS components. A fuzzy cost-reliability trade-off optimization model of COTS selection in the development of a modular software system is developed. The proposed fuzzy optimization model minimizes the total cost of the software system, satisfying the constraints of minimum threshold on system reliability, maximum threshold on the delivery time of the software, and incompatibility among COTS components. The model considers uncertainty in the coefficients of the objective function and constraints. In general, the fuzzy data in fuzzy mathematical programming are expressed by using fuzzy numbers. To deal with uncertainty in real-world applications of COTS selection, in this chapter, the coefficients of the cost objective function, delivery time constraints, and minimum threshold on reliability are considered fuzzy numbers.

For the software system considered in this chapter, let $\tilde{C}_{ij}$ be the imprecise cost of the $j$-th COTS component for the $i$-th module, $\tilde{D}_{ij}$ be the imprecise delivery time of the $j$-th COTS component for the $i$-th module, $\tilde{T}$ be the imprecise overall delivery time, and $\tilde{Z}$ be the fuzzy total cost. Furthermore, let $x_{ij}$ be the binary variable indicating whether the $j$-th COTS component of the $i$-th module is chosen or not. Also, $s_i$ is the average number of invocations of the $i$-th module, $p_i$ is the probability of occurrence of at least one failure when the $i$-th module is executed, $\mu_{ij}$ is the probability of failure on demand of the $j$-th COTS component in the $i$-th module, and $\tilde{R} = (\frac{1}{\tilde{R}^*} - \tilde{1})$ where $\tilde{R}^*$ is the imprecise minimum system reliability threshold. To consider the incompatibility among COTS components, we use contingent decision (either-or) constraints. Because the presence of the either-or constraints makes the optimization problem nonlinear, it is linearized by introducing a binary variable $y_i$ where $y_i = 0$ if the $i$-th constraint is active; otherwise, $y_i = 1$. The fuzzy optimization model for COTS selection used in this chapter is formulated as follows:
In contrast to most of the previous literature on fuzzy COTS selection, we develop a solution approach by using Zadeh’s extension principle to provide fuzzy solutions instead of a crisp solution of the fuzzy optimization model. The fuzzy optimization model \( P(1.15) \) is converted into a pair of mathematical programming problems parameterized by possibility (feasibility) level \( \alpha \) by using Zadeh’s extension principle. The membership function of the fuzzy total cost is obtained by transforming the fuzzy model into two conventional crisp mathematical programming models that provide the lower and upper bounds of the fuzzy objective value at different possibility (feasibility) levels \( \alpha \). The lower and upper bounds of the fuzzy objective value at different possibility levels \( \alpha \) generate the left-shape function and the right-shape function, respectively, of the membership function of the fuzzy cost. The solution approach provides the DM enough flexibility in maintaining cost-reliability trade-off of COTS selection besides meeting other important system requirements. A real-world case study is also discussed to demonstrate the effectiveness of the proposed model in a fuzzy environment.
In Chapter 3, we again assume that modular technique is used for software development. The software system performs one or more functions, as is the requirement. A program performs each function, and the probabilities of execution of various programs are known. Each program consists of a number of modules, and the software system contains a total of \( m \) modules that are executed sequentially. Some of these modules may be common for different programs. Also, we assume that \( n_i \) alternative COTS components developed by different groups are available for implementing the \( i \)-th module. Each module in a software system has different levels of importance that depend on access frequency. Users often recognize the quality of a software system on the basis of the quality of the modules; hence, purchasing expensive high-quality COTS components may be justified by the frequent use of the module. In this chapter, we assign different weights to the modules according to their access frequencies and also by using software developer’s inputs regarding technical specifications of the software system. The weight \( w_i \) assigned to the \( i \)-th module is calculated by using AHP. Furthermore, let the purchasing cost \( c_{ij} \) of the \( j \)-th COTS component for the \( i \)-th module be provided by the vendor and \( q_{ij} \), which represents the quality level of the \( j \)-th COTS component for the \( i \)-th module, is estimated by using the ISO/IEC 9126 quality model. This chapter is divided into two sections.

In the first section, we formulate fuzzy biobjective optimization models for COTS selection on the basis of vague aspiration levels of the DM in respect to the weighted quality and the cost of COTS components. The models are constrained by several limitations including the incompatibility among COTS components and the selection of one and only one COTS component for each module. The biobjective optimization model for COTS selection is formulated as follows:

\[
\begin{align*}
\text{P}(1.16) \quad & \max Q(x) = \sum_{i=1}^{m} w_i \left( \sum_{j=1}^{n_i} q_{ij} x_{ij} \right) \\
& \min C(x) = \sum_{i=1}^{m} \sum_{j=1}^{n_i} c_{ij} x_{ij} \\
& \text{subject to} \\
& \sum_{j=1}^{n_i} x_{ij} = 1, \ i = 1, 2, \ldots, m, \quad (1.1)
\end{align*}
\]
\[ x_{rs} - x_{ut_k} \leq My_k, \quad k = 1, 2, \ldots, z, \tag{1.2} \]
\[ \sum_{k=1}^{z} y_k = z - 1, \tag{1.3} \]
\[ y_k \in \{0, 1\}, \quad k = 1, 2, \ldots, z, \tag{1.4} \]
\[ x_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n_i. \tag{1.5} \]

In the literature, various types of membership functions, each having its own distinct advantages, have been proposed to express vague aspiration levels of the DM. In this section, a nonlinear S-shape membership function is used to represent the vague aspiration levels of the DM defined by

\[ f(x) = \frac{1}{1 + \exp(-\alpha x)}, \]

where \( \alpha \), \( 0 < \alpha < \infty \), is a fuzzy parameter that measures the degree of vagueness. We use Bellman-Zadeh’s maximization principle (the max-min approach) to formulate a fuzzy biobjective optimization model under the assumption that both the fuzzy objectives are treated equivalently. To reflect the relative importance of the DM for the two objectives, a weighted additive fuzzy optimization model is also presented in which different weights are assigned to the two objectives. Furthermore, the efficiency of the obtained solutions is verified by using a two-phase approach. The two-phase approach guarantees a fuzzy-efficient solution and improves, if possible, the solution obtained by using the max-min approach. The main advantage of the proposed models is that if the DM is not satisfied with any of the COTS selection obtained, more COTS selections can be generated by varying values of shape parameters of nonlinear S-shape membership functions. With the help of numerical illustrations, we demonstrate the efficiency of the proposed models.

In the second section, we extend the fuzzy biobjective optimization model of the first section to a fuzzy multiobjective optimization model. The fuzzy optimization model minimizes the cost, development efforts (number of lines of code), and execution time and maximizes the reliability and weighted quality of the software system. Furthermore, the model is subjected to many realistic constraints including incompatibility among COTS components, delivery time constraints, and the selection of one and only one COTS component for each
1.4 Structure of the thesis

module. In addition to the notations used in the formulation of model $P(1.16)$, let $\nu_p$ denote the probability of use of the $p$-th program, $s_i$ denote the average number of invocations of the $i$-th module, $S_p$ the set of modules corresponding to the program $p$, $l_{ij}$ the size (number of lines of code), $t_{ij}$ the expected execution time, $\mu_{ij}$ the probability of failure on demand, and $d_{ij}$ the delivery time of the $j$-th COTS component for the $i$-th module. The multiobjective optimization model for COTS selection is formulated as follows:

$$
\begin{align*}
\text{max } Q(x) &= \sum_{i=1}^{m} w_i \left( \sum_{j=1}^{n_i} q_{ij} x_{ij} \right) \\
\text{min } C(x) &= \sum_{i=1}^{m} \sum_{j=1}^{n_i} c_{ij} x_{ij} \\
\text{max } R(x) &= \prod_{i=1}^{m} \exp \left( -s_i \sum_{j=1}^{n_i} \mu_{ij} x_{ij} \right) \\
\text{min } L(x) &= \sum_{i=1}^{m} \sum_{j=1}^{n_i} l_{ij} x_{ij} \\
\text{min } T_e(x) &= \sum_{p=1}^{P} \nu_p \sum_{i \in S_p} \left( \sum_{j=1}^{n_i} l_{ij} x_{ij} \right)
\end{align*}
$$

subject to

$$
\sum_{j=1}^{n_i} d_{ij} x_{ij} \leq T, \ i = 1, 2, \ldots, m,
$$

and Constraints (1.1)–(1.5).

Here, we use linear membership functions to describe the fuzzy goals of the multiobjective optimization model. By using Bellman-Zadeh’s maximization principle and linear membership functions, we formulate the fuzzy optimization model by giving equal importance to all objectives. Furthermore, to determine the preferred compromise solution for the fuzzy optimization model, we use an interactive approach. To satisfy the individual preferences of the DM for various objectives, in this approach, we improve the obtained compromise solution by modifying the lower (upper) bound and the aspiration level(s) of the selected objective(s). As desired by the DM, an individual objective(s) can be improved but because of the multiobjective nature of the problem, this improvement
can produce adverse effects on the other objective(s). Hence, depending on the preferences of the DM for the various objectives, we can modify the compromise solution and this process is continued until the DM terminates the process; that is, the preferred compromise solution is obtained or no further improvement is made. Numerical illustrations are provided to demonstrate the usefulness of the proposed model and the solution approach.

Chapter 4, in contrast to the previous two chapters, deals with decision making situations in which a software developer concurrently undertakes the development of $N$ applications for $M$ modules such that the $i$-th application requires $m_i$ modules. It is assumed that each module belongs to an application uniquely and there is no module that is common to different applications. Generally, in CBSD, it is considered that COTS components that belong to a set of alternative COTS components have similar functionality, but, in this chapter, we consider the functional contributions of COTS components toward functional requirements of a component-based software system for COTS selection process. We distribute all the $L$ COTS components available in COTS components’ market into $T$ number of sets of alternative COTS components, such that $s_t$ denotes the set of alternative COTS components that fulfills the $t$-th functional requirement of the software system. Also, the functional rating of the $k$-th COTS component to the $j$-th module is given by $f_{jk}$. Furthermore, for a good software system we desire modules with high cohesion and low coupling.

Considering that the relationship between cohesion and coupling of modules can be measured by using $ICD$, we assign a minimum threshold value of $ICD$, that is, $H_i$ for each of the $i$-th application. We also consider the reusability of components in different applications. This chapter is divided into two sections.

In the first section, we propose a biobjective optimization model that maximizes the functional requirements of the modular software system and minimizes the total development cost of the software system. The total development cost includes the procurement and adaptation costs of COTS components. The model is constrained by several realistic constraints including a minimum threshold on the $ICD$ for each application, reusability of COTS components, selection of one and only one COTS component from a set of alternative components for each functional requirement per application, and the selection of more than
The biobjective optimization model for COTS selection is formulated as follows:

\[
\text{P(1.18)} \quad \begin{align*}
\text{max } F(x) &= \sum_{j=1}^{M} \sum_{k=1}^{L} f_{jk} x_{j,k} \\
\text{min } C(x) &= \left( \sum_{k=1}^{L} c_p^k y_k + \sum_{j=1}^{M} \sum_{k=1}^{L} c_a^j x_{j,k} \right)
\end{align*}
\]

subject to

\[
\sum_{j=1}^{M} s_{ij} \left( \sum_{k=1}^{L-1} \sum_{k'=k+1}^{L} r_{kk'} x_{j,k} x_{j,k'} \right) + \sum_{k=1}^{L-1} \sum_{k'=k+1}^{L} r_{kk'} \left( \sum_{j=1}^{M} s_{ij} x_{j,k} \right) \left( \sum_{j=1}^{M} s_{ij} x_{j,k'} \right) \geq H_i,
\]

\[i = 1, 2, \ldots, N, \tag{1.6}\]

\[\sum_{j=1}^{M} s_{ij} x_{j,k} \leq 1, \quad i = 1, 2, \ldots, N, \quad k = 1, 2, \ldots, L, \tag{1.7}\]

\[\sum_{j=1}^{M} x_{j,k} \leq y_k \cdot N, \quad k = 1, 2, \ldots, L, \tag{1.8}\]

\[x_{j,k} \leq b_{jk}, \quad j = 1, 2, \ldots, M, \quad k = 1, 2, \ldots, L, \tag{1.9}\]

\[\sum_{k \in s_t} \sum_{j=1}^{M} s_{ij} x_{j,k} = 1, \quad i = 1, 2, \ldots, N, \quad t = 1, 2, \ldots, T, \tag{1.10}\]

\[\sum_{k=1}^{L} x_{j,k} \geq 1, \quad j = 1, 2, \ldots, M, \tag{1.11}\]

\[x_{j,k}, y_k \in \{0, 1\}, \quad j = 1, 2, \ldots, M, \quad k = 1, 2, \ldots, L, \tag{1.12}\]

where \(c_p^k\) and \(c_a^j\) denote the purchasing price and the adaptation cost of the \(k\)-th COTS component, respectively, \(r_{kk'}\) gives the number of interactions between \(k\)-th and \(k'\)-th COTS components, \(s_{ij}\) is the binary parameter that shows whether the \(j\)-th module belongs to the \(i\)-th application or not, and \(b_{jk}\) denotes whether the \(k\)-th COTS component can be reused to achieve the \(j\)-th module or not.

We use the weighted sum method to solve the biobjective optimization problem. The model sensitivity has been shown with respect to changes in the minimum threshold value of the ICD for each application and also by varying the weight.
parameters of the two objective functions reflecting the preferences of the software development team (here the DM). A real-world scenario of developing two financial applications for two small-scale industries is included to demonstrate the efficiency of the model.

In the COTS selection optimization model $P(1.18)$, we require an estimate of functional ratings and cost of COTS components. The exact calculation of these estimates may not be possible because DM’s assessment about these estimates may be based on incomplete knowledge about COTS components itself and other aspects including vendor’s credentials. Under such circumstances, the issue of selecting COTS components becomes an issue of a choice from a ‘fuzzy’ set of subjective/intuitive interpretations. Therefore, we formulate fuzzy biobjective optimization models for COTS selection based on vague aspiration levels of the DM. The fuzzy biobjective optimization models proposed in the second section of this chapter are an extension of the optimization model proposed in the first section. The proposed models maximize the functional performance and minimize the total cost of the software system, satisfying the constraints of minimum threshold on $ICD$ and reusability of COTS components. We use nonlinear S-shape membership functions to express vague aspiration levels of the DM regarding functional requirements and the total development cost of the modular software system. Following Bellman-Zadeh’s maximization principle (the max-min approach) and using the S-shape membership functions, the fuzzy optimization model for COTS selection based on cohesion and coupling under multiple applications environment is formulated as follows:

$$\begin{align*}
    P(1.19) \quad & \max \lambda \\
    \text{subject to} & \quad \frac{1}{1 + \exp\left(-\alpha_F \left(\sum_{j=1}^{M} \sum_{k=1}^{L} f_{jk}x_{j,k} - F_m\right)\right)} \geq \lambda, \\
               & \quad \frac{1}{1 + \exp\left(\alpha_C \left(\sum_{k=1}^{L} c_k y_k + \sum_{j=1}^{M} \sum_{k=1}^{L} c_{jk} x_{j,k} - C_m\right)\right)} \geq \lambda, \\
               & \quad 0 \leq \lambda \leq 1, \\
               & \quad \text{and Constraints (1.6)--(1.12),}
\end{align*}$$

where $F_m$, $C_m$ are the midpoints (middle aspiration level) at which the membership function value is 0.5 and $\alpha_F$, $\alpha_C$ are shape parameters that may be
given by the DM on the basis of his/her own degree of satisfaction. Furthermore, to reflect the relative importance of the DM for the two objectives, a weighted additive fuzzy optimization model is solved in which different weights are assigned to the two objectives. The efficiency of the obtained solutions is verified by using a two-phase approach. The applicability of the model in real-world situations is demonstrated through a case study. The main advantage of the proposed models is that if the DM is not satisfied with any of the COTS selection obtained, more COTS selections can be generated by varying values of shape parameters of nonlinear S-shape membership functions.

In Chapter 5, we consider the build-or-buy decision under multiple applications development task; that is, we assume that for the development of modular software systems, a few components can be developed in-house along with COTS components. There can be more than one alternative COTS component in COTS components’ market to fulfill the requirements of a specified module, but there is only one in-house development corresponding to the modules having similar functionalities in all the applications. It is assumed here that each module will be fulfilled through selecting and adapting only one component either from the available COTS components or from in-house development. We develop a nonlinear cost/quality optimization model that supports the decision whether to buy COTS components or to build in-house components to minimize the overall cost of the software system while maintaining satisfactory values of various quality attributes and considering reusability and compatibility simultaneously. Quality constraints are related to the delivery time and the reliability of the applications. The proposed model assists software developers in selecting software components when multiple applications are undertaken concurrently and provides the amount of unit testing to be performed on in-house developed components. The optimization model for component selection is formulated as follows:
\[ P(1.20) \quad \min C(x) = \left( \sum_{k=1}^{L} c_k^o y_k + \sum_{k=1}^{L} \sum_{j=1}^{M} c_{j,k}^a x_{j,k} + \sum_{j'=1}^{J} c_{j'}^p (t_{j'} + \tau_{j'} N_{j'}^{tot}) z_{j'} + \right. \\
\left. \sum_{j'=1}^{J} \sum_{j} (c_{j,j'}^o z_{j,j'}) \right) \\
\text{subject to} \\
\sum_{j=1}^{M} s_{ij} x_{j,k} \leq 1, \; i = 1, 2, \ldots, N, \; k = 1, 2, \ldots, L, \\
\sum_{j=1}^{M} x_{j,k} \leq y_k \cdot N, \; k = 1, 2, \ldots, L, \\
x_{j,k} \leq s_{jk}, \; j = 1, 2, \ldots, M, \; k = 1, 2, \ldots, L, \\
\sum_{j=1}^{M} s_{ij} z_{j,j'} \leq 1, \; i = 1, 2, \ldots, N, \; j' = 1, 2, \ldots, J, \\
\sum_{j=1}^{M} z_{j,j'} \leq z_{j'} \cdot N, \; j' = 1, 2, \ldots, J, \\
z_{j,j'} \leq s_{j,j'}, \; j = 1, 2, \ldots, M, \; j' = 1, 2, \ldots, J, \\
\prod_{j=1}^{M} s_{ij} \exp \left( - \left( \sum_{j'=1}^{J} (1 - \rho_{j'}) s_{j,j'} + \sum_{k=1}^{L} s_{j,k} y_k \nu_k \right) \right) \geq R_i, \\
i = 1, 2, \ldots, N, \\
\sum_{k'=k+1}^{L} \sum_{j=1}^{L} u_{kk'} \left( \sum_{j=1}^{M} s_{ij} x_{j,k} \right) \left( \sum_{j=1}^{M} s_{ij} x_{j,k'} \right) + \sum_{j'=1}^{J} \sum_{j''=j'+1}^{J} u_{j'j''} \left( \sum_{j=1}^{M} s_{ij} z_{j,j''} \right) \\
\left( \sum_{j=1}^{M} s_{ij} z_{j,j''} \right) + \sum_{j'=1}^{J} \sum_{k=1}^{L} u_{kk'} \left( \sum_{j=1}^{M} s_{ij} x_{j,k} \right) \left( \sum_{j=1}^{M} s_{ij} z_{j,j'} \right) = \left( \sum_{j=1}^{M} s_{ij} \right), \\
i = 1, 2, \ldots, N, \\
\sum_{j'=1}^{J} (t_{j'} + \tau_{j'} N_{j'}^{tot}) z_{j,j'} + \sum_{k=1}^{L} x_{j,k} d_k \leq T, \; j = 1, 2, \ldots, M, \\
\sum_{j'=1}^{J} z_{j,j'} + \sum_{k=1}^{L} x_{j,k} = 1, \; j = 1, 2, \ldots, M, \\
x_{j,k}, y_k, z_{j,j'}, z_{j'} \in \{0, 1\}, \; j = 1, 2, \ldots, M, \; k = 1, 2, \ldots, L, \; j' = 1, 2, \ldots, J.
Here, $J$ denotes the number of distinct in-house components that can be developed for the software system, $d_k$ the delivery time, and $\nu_k$ the probability of failure on demand of the $k$-th COTS component. Furthermore, for $j'$-th component developed in-house, $c_{j'}$ denotes the unitary development cost, $c^{a}_{jj'}$ the adaptation cost, $t_{j'}$ the estimated development time, $\tau_{j'}$ the average time required to perform the test case and $\pi_{j'}$ the testability. Also, the set of binary parameters for both COTS and in-house developed components includes the following: $s_{ij}$ denotes whether the $j$-th module belongs to the $i$-th application or not, $s_{jk}$ whether the $k$-th COTS component can be reused to implement the $j$-th module or not, and $s_{jj'}$ whether the $j'$-th in-house developed component can be reused to implement the $j$-th module or not. In addition, compatibility parameters used to design the model include the following: $u_{kk'}$ denotes whether or not the $k$-th COTS component is compatible with the $k'$-th COTS component; $u_{kj'}$ whether the $k$-th COTS component of one module is compatible with the $j'$-th in-house component developed for other module; and if there are no compatibility issues between COTS components and in-house developed components, then $u_{kj'} = 1$. We have also used the following binary variables to implement the build-or-buy decision strategy: $x_{jk}$ indicates whether the COTS component is selected to implement the $j$-th module or not, $y_k$ indicates whether the $k$-th COTS component is selected or not, $z_{jj'}$ indicates whether the $j'$-th in-house developed component is selected to implement the $j$-th module or not, and $z_{j'}$ indicates whether the $j'$-th component is developed in-house or not.

A small-scale case study in which a software developer undertakes two financial applications for two small-size companies is used to demonstrate the efficiency of the model P(1.20) in real-world applications. A model sensitivity with respect to changes in the component characteristics and constraint thresholds is performed to reflect changes in the component selection corresponding to changes in the input parameters.