A PYTHON PACKAGE FOR SOLVING NONLINEAR ADDITIVELY SEPARABLE NONCONVEX OPTIMIZATION PROBLEMS

Ishanga Chathurika Udatiyawala

(189354H)

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

May 2021

A PYTHON PACKAGE FOR SOLVING NONLINEAR ADDITIVELY SEPARABLE NONCONVEX OPTIMIZATION PROBLEMS

Ishanga Chathurika Udatiyawala

(189354H)

Dissertation submitted in partial fulfillment of the requirements for the degree Master of Science in Computer Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

May 2021

Candidate's Declaration

I hereby declare that this report was carried out by me under the supervision of Dr. Charith Chitraranjan, in the Department of Computer Science and Engineering, University of Moratuwa as a partial fulfillment of the requirements of the of the M.Sc. degree in Computer Science at the University of Moratuwa. It has not been submitted to any other institution or study program by me for any other purpose.

Candidate's name : Ms. I.C. Udatiyawala

UOM Verified Signature

Condidate's signature: …….....

Date : …31.05.2021……

Supervisor's Recommendation

I/We certify that the above declaration made by the candidate is true & this report is forwarded for the purpose of evaluation.

UOM Verified Signature

……

Dr. C. Chitraranjan

Senior Lecturer,

Department of Computer Science and Engineering,

University of Moratuwa,

Katubadda.

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my supervisor Dr. Charith Chitraranjan, Senior Lecture, Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka, who have given me immense support, encouragement and excellent guidance throughout project. His motivation, persuasion and patience helped me to overcome many crisis situations and complete this research successfully.

I convey my sincere gratitude to member of family who were the source of encouragement behind me and all of other who encouraged me the correct path of my success. My sincere thanks go for all my friends who helped me in many ways to make this project success.

ABSTRACT

Nonlinear optimization problems are difficult to solve in many practical situations, and only a few approaches are developed in recent history. The main problem in the nonlinear optimization problem is it needs some convexity approximation to obtain global optima. Otherwise, it ends up with the local optima. Separable programming is a common method to solve nonlinear optimization problems. Most of the nonlinear programming techniques, including the separable programming end up with the local optima when the problem is nonconvex. To overcome this problem several types of research have been conducted in recent history.

Computational implementations are an essential component in nonlinear programming as they are hard to solve with traditional methods. Also, very limited computational implementations have been developed in this area. However, in general, such implementations cannot promise that the results they generate are globally optimal. Therefore "**laptimize**" python package was developed to solve nonlinear separable nonconvex optimization problems using a branch and bound method. This algorithm was proposed by James E. Falk of The George Washington University. It has been verified that it can produce a global solution in a finite number of steps for a large class of nonlinear programming problems in the separable class. The correctness, stability, and convergence of the algorithm are evaluated with nonlinear programming examples.

Table of Contents

Table of Figures

Table of Tables

CHAPTER 01

INTRODUCTION

1.1 Overview

This chapter is dedicated to give an introduction of the study. Section 1.2 explains the general background of the study and the objectives of the study are mentioned under section 1.3. A brief introduction about the terms used in this study is described in section 1.4. The significance of the study is described in section 1.5.

1.2 Background of the study

Mathematical programming is a key tool in operations research, and it's widely used to solve optimization problems in a variety of fields. In the late 1940s, linear programming and the simplex method were introduced as the first step in optimization history, followed by convex optimization and the interior point method in the late 1980s. As more people understand the formulations in different applications, the usefulness of linear programming has been accompanied by a transforming phase of application cycle. More analysis has been done on the theory, algorithms, and software. As the tool has become more effective, more people are appreciating its use.

However, in recent history many importance problems cannot be easily modeled in a linear context. Nonlinear programming is used in missile allocation, failure diagnosis, media selection for ads, facility location, chemical process scheduling, sewer design, and other applications, according to a study of recent literature. Nonlinear optimization problems must be solved for these types of analyses.

The main problem in nonlinear optimization problem is its need some convexity approximation to obtain global optima. Otherwise, it is end up with the local optima. Several kinds of research have been conducted to overcome this problem using mathematical algorithms such as branch and bound. Most common method to solve nonlinear optimization problems is separable programming which can be apply for piecewise linear problems. In objective and constraints, separable programming effectively replaces all separable functions with piecewise linear functions.

According to the definition of the separable programming, Convex programs can be defined as nonlinear programming algorithms that maximize or minimize a convex function over convex constraints [1].

Further convex program can be solved using separable programming technique with global optima. But if NLP is non-convex program, still the concept of a separable program can be applied to it. However, in such situations, the best solution can be a local rather than a global one.

The standers method for treating separable form have two steps:

- Approximating problem by piecewise linear functions
- Appling restriction basis entry rule to simplex method when solving the problem

The modification method will result a local but not necessarily a global solution of the approximating problem.

1.3 Objectives

The objectives of the study are;

- Develop a general python package to solve nonlinear optimization problems which are separable and non-convex.
- Compare results with existing nonlinear optimization problem solving packages

1.4. Optimization

Optimization is a common mathematical method that refers to a set of mathematical concepts and methods. Also, optimization become a core component in data science as well. It can be applied to physics, biology, engineering, economics, and industry to solve quantitative problems. After the 1940s, computer programming came into the programming field. Before that, the term mathematical programming was used. At the same time, optimization methods have developed with an acceleration in the field of operations research, mathematical economic science, game theory, numerical analysis, management theory, computer science, and combinatorics.

The most basic aspect of optimization is minimizing or maximizing an objective function, which can be defined as a single numerical value, or limitations on the values that variables can take is the second component of an optimization problem. Linear programming and Nonlinear Programming are the main mechanisms of optimization. Any given optimization problem can be identified as one of them.

The term "linear programming" refers to a main method of optimization. According to the definition of linear programming objective function or constraints cannot be consists of the power of variables such as squares roots, squares, or products of the variables. Most of the fundamentals of optimization have buildup with the linear programming concept as it is the simplest as well as the powerful technique in the operation research area.

Nonlinear programming is another essential form of optimization. According to the definition of nonlinear programming objective function or constraints can consists of the power of variables such as squares roots, squares, or products of the variables. If at least one objective or constraint includes a nonlinear term, the problem is identified as a nonlinear problem. Nonlinear problems are harder to solve for the global solution.

In this research mainly focusing on the non-linear optimization problems. There are several methods to solve the nonlinear optimization problems and separable programming is a one of them.

1.4.1 Linear Programing

In recent past, linear programming is a commonly used mathematical programming technique for solving everyday decision problems. It was first proposed in 1947 by Joseph Fourier, a French mathematician. It was relatively straight forward mathematically, and the approach's ground breaking aspect is that it expresses the decision-making aim in terms of minimizing or maximizing a linear objective function with respect to the linear constraint.

In 1947, the simplex approach was introduced, which proved to be effective in solving practical problems. The general simplex method was developed in 1951 for the SEAC computer of the United States Bureau of Standards. Recently, as result of massive improvement in data science people starts to build special structures to solve massive

linear problems. With the help of computer science, lot of computational approaches are available for solving linear programming

1.4.2 Simplex Method

The most popular and earliest method for solving linear programming problems is the simplex method. The simplex method was developed by George Dantzig, a mathematical adviser for the United States Air Force, in 1947. Hereafter most of the operational research related problems were solved using simplex method. When objective function and constraints are linear, this approach is the most effective and useful algorithm for solving optimization problems. Furthermore, this algorithm has been used as a basic approach in the creation of most computer programs.

1.4.3 Nonlinear Programing

While the linear programming model works well in many cases, some problems need nonlinear components to be correctly modeled. Efforts to efficiently solve such nonlinear problems have progressed rapidly over the last three decades. The problem of optimizing an objective function or constraints that consists of the power of variables such as squares roots, squares, or products of the variables needs to be solved by non-linear programming

Convex analysis, duality theory, and control theory were all major inspirations on the growth of nonlinear programming after 1940. Convex analysis is a major component of nonlinear programming which study of convex sets and convex functions. Prescence of convexity make life easier when solving nonlinear programming problems. Nonlinear programming problems are hard to solve when objective function or constraints are nonconvex [1].

Sequential quadratic programming, integer programming, separable programming, and fractional programming are all important algorithmic approaches in nonlinear programming. Above methods are the same of important sub topics of nonlinear optimization models. Separable programming is a special solving method to deal with nonlinear optimization problems.

1.4.3 Separable Programing

The method of separable programming was initially formulated by Miller (1963). Separable programming is significant because it enables a linear programming model to approximate a convex nonlinear program with arbitrary accuracy. There are two types of separable programming concept, those are convex and concave. Convex separable function in which the objective and restriction functions are both convex.

The main idea of separable programming is replacing the original separable problem with a piecewise linear approximation. As a result, using the simplex approach with the restricted basis entry law, a global solution can be obtained.

The main problem arises when the objective function and/or constrains are nonconvex. In this case, either a mixed-integer linear programming problem must be solved, or the model can be solved directly using a modified version of the simplex algorithm with a limited basis entry rule. Although a local optimum is obtained in this situation, the global optimum can be discovered using the branch and bound process [2].

1.4.4 Branch and Bound Algorithm

Branch and bound is a type of algorithm that is commonly used to solve problems involving combinatorial optimization. In the worst-case scenario, these problems are exponential in terms of time complexity and may necessitate exploring all possible permutations. These problems are easily solved with Branch and Bound. The principle behind this approach is that the entire set of feasible solutions can be partitioned into smaller subsets. The best solution can then be sought by systematically evaluating these smaller subsets.

1.5 Significance of the Study

Several computer codes have been developed in recent history to solve linear and nonlinear optimization problems and some of them are publicly available. Python also has some packages to solve nonlinear optimization problems which are publicly available (Scipy.optimize and Mystic). However, in general, such codes cannot promise that the results they generate are globally optimal. Further, many of

programming languages don’t have implemented package for separable nonconvex programming.

Therefore, developing a package for solving nonconvex separable problems is noteworthy for future nonlinear optimization researches.

CHAPTER 02
REVIEW OF LITERATURE

Initially, an algorithm to obtain global solution for nonconvex optimization problems was developed in 1969 by Falk and Soland [2]. This algorithm is based on the branch and bound method and solves a sequence of problems in which each of the objective function is convex. Also, they have discussed several examples and some computational considerations. As they mentioned in research paper this algorithm is very similar to a process briefly suggested by Lowler and Wood [3]. In here they only consider the problems of the form: Find $x = (x1, \dots., xn)$ to minimize $\sum \varphi_i(x_i)$ subject to $x \, \varepsilon \, G$ and $l \leq x \leq L$, each φ_i is assumed to be lower semicontinuous, possibly nonconvex and G is assumed to be closed. All of the algorithms which are proposed early to this, in which convergence to an optimal solution is assured. Because only a finite number of solutions exist. Limitation of this algorithm is, in general the algorithm does not converge in a finite number of steps. But it converges under different conditions. Further, they specifically point out that They only consider the separability of the objective function. Additionally, they have used two different examples to proof convergence of the algorithm and graphical illustration as well. One of these examples consist of concave objective function and linear constraints, while second one is neither convex nor concave.

In the second version of this algorithm was published by Soland [4] which extend the previous version of the above algorithm, additionally including the nonconvex constraints. In this publication they consider the problems of the form: Find $x = (x1, \dots., xn)$ to minimize $\sum \varphi_i(x_i)$ subject to $x \, \varepsilon \, G$ and $l \leq x \leq L$ and $\sum \varphi_{ij}(x_i) \leq 0, j = 1, \dots, m,$ each φ_{ij} is assumed to be lower semi continuous, possibly nonconvex and G is assumed to be closed. According to Beale [3] the piecewise linear functions are used to approximate the constraints and objective functions when solving separable programming problems. But in this paper, they represented them as mixed integer programming problems. But in this solution, they allow the functions defining G to be non-separable.

The algorithm which will be used for this general implementation is based on the algorithm proposed by Flank [5]. This algorithm was based on branch and bound method it was mention that algorithm gives the global solution in finite number of

steps. In Flank paper, there is another implementation which was developed by Beale and Tomlin [6] and being developed from an integer programming point of view while this has modifications of the rules developed in the [5]and its extension by Soland [7].

This algorithm was built on top of the linear programming concepts, since it uses branch and bound method to come up with the sequence of linear programming sub problems. By using appendices of [5] algorithm and Paul linear programming subroutine, computer program for solve the non-convex optimization problem was developed by Grotte [8]. In this paper indicate that, the code is significant in that it seeks a global optimum in a finite number of steps and has been shown to be robust when applied to large problems. The size of problems that can be handled is only restricted by computer storage and run time considerations, as this code has been tested on a wide variety of problems. The code mentioned in this paper, which is implemented in the MOGG program, was designed to be stable and accurate while also addressing some minor computational inefficiencies.

Before above implementation there were several researches conducted by many researchers related to separable programming, branch and bound method and nonlinear optimization problems. Also, there were researches by indicating consideration for computational implementation as well.

Lawler and Woods [3] conducted a survey about branch and bound methods. The important features and several specific applications were discussed in this survey that are related to the branch and bound approach to constrained optimization. Integer linear programming, nonlinear programming, and the quadratic assignment problem have been described in this paper. Additionally, computational issues were addressed, including trade-offs between computation length and storage requirements, as well as a reference to dynamic programming. Various applications not related to mathematical programming were also listed.

After that limited number of researchers have research about this research area and only have few numbers of research materials. Some of the use scenarios of the algorithm as follows.

A method to find the global optimum for nonconvex separable problems using mixed integer programming was introduced in 1999 [9]. Mixed integer programming is one

of main method to solve nonlinear problems. This method requires large number of 0 and 1 variables to solve nonconvex problems. In this paper, they have proposed a way to use a smaller number of 0 and 1 variables to come up with approximate global solution for nonlinear separable programming. Separable objective and constraint functions are first famulated as a piecewise linear function with absolute term summation. Then they have converted nonlinear separable problem to mixed 0-1 problem which is solvable and approximated solution for the original problem.

In [10] have suggested two new deterministic global optimization algorithms for nonconvex mixed-integer problems. Both algorithms are built using branch-and-bound principles, but they take different approaches to each of the necessary steps. This special structure mixed-integer algorithm, tackles problems with nonconvexities in continuous variables and linear and mixed-bilinear involvement of binary variables. The general structure mixed-integer algorithm works for a wide variety of problems where the continuous relaxation is twice continuously differentiable.

Lu and Ito [11] presented a method to use feedforward neural networks to convert nonlinear programming problems into separable programming problems. This is a new research area which is develop on top of the separable programming and can be useful when developing separable programming problems. The method was designed to take advantage of two main features of use feedforward neural networks, their ability to accurately approximate arbitrary continuous nonlinear functions and their ability to convert nonlinear functions as parameterized compositions functions of single variables. Any non separable objective functions as well as the constraints in nonlinear programming problems can be approximated as separable functions with use feedforward neural networks based on these two characteristics. As a result, every nonlinear programming problem can be transformed into a separable programming problem. This paper also provided several examples to illustrate the process and results of the simplex method for solving SP problems using the restricted basis entry rule.

In 2004 [12], a new branch and bound method have been proposed for solving large scale nonlinear concave separable problems. In here, branching was done by using largest distance bisection method which has proven that is a normal rectangle subdivision. This new subdivision method uses to divide rectangle into sub rectangles when branching is needed. One large scaled numerical example has used to test the

efficiency of the proposed algorithm. It has been proven that his branch and bound method can coverage effectively for large scale problems.

Further, in 2004 Kuno and Shi [13] proved that Flank and Soland [2] branch and bound method is an important to solve concave separable problems. They have developed two algorithms to globally solve the linear programming problems with a concave constraint. These are special class of linear programming problems. Flank and Soland [2] branch and bound method was applied to the problem in direct and indirect ways. The problem of branch-and-bound and alternating local search was solved in the direct application. They used a parametric right-hand-side simplex algorithm to perform the bounding operation in the indirect application

Croxton and Gendron [14] identified that linear programming-based branch and bound method and Falk and Soland [2] compute the same lower bounds for convex envelop. In order to prove this, a generic minimization problem was investigated with separable nonconvex piecewise linear costs. Also, results were used to get a connection between Lagrangian duality and linear programming.

Stuber [15] conduced a research to introduce a differentiable model for optimizing hybridization of industrial process heat systems with concentrating solar thermal power. In here, they used convex and nonconvex formulations to get the solution. Branch and Bound separable programming were used to solve the nonconvex optimization problem.

According to the literatures that have found no one ever has develop a python package to solve separable non-convex optimization problems in recent history. Large number researches have been developed under this research area, but all of those are limited to theoretical implementations. In most of the cases separable non-convex optimization problems have been limited to mixed-integer frame work since it is easier than this approach.

Also, most of the literatures have proven that Flank and Soland [2] branch and bound method is a useful method to solve nonconvex separable optimization problems. In this research, we are using the Flak [5] extension of this approach which if more convenient to generalize and implement.

CHAPTER 03
THEORETICAL BACKGROUND OF THE STUDY

3.1 Overview

This section gives the brief introduction about the algorithm which is going use for package development. This is a summary of algorithm by Flank [5]. Original paper has attached in appendix A. In section 3.2 describes the problem form and section 3.3 about problem specification.

3.2 Introduction to the Algorithm

Proposed python package will be based on algorithm developed by the Flank [5]. Theoretical background of the separable programming and algorithm as follows.

The optimization problem which we are considering has the form of

$$\text{Problem P} \begin{cases} minimize \quad F_0(X) \\ subject\ to\ F_i(X) \leq b_i \quad i = 1,2,\dots\dots\dots,m \\ l \leq X \leq L \end{cases}$$

Where l and L are the finite lower and upper bounds of on x respectively.

Let's assume that each $F_i \quad i = 1,2,\dots\dots m$ is separable i.e

$$F_i(X) = \sum_{j=1}^{n} F_{ij}(x_j) \qquad i = 1,2,\dots\dots\dots.m$$

And that each F_{ij} is continuous.

This type of problems famulated in many fields such as logistics, econometric data fitting, cost optimization, capacity optimization. and statistics [1].

In the next step one can formulate a new problem by replacing each nonlinear function by an approximating piecewise linear function. These piecewise linear functions are called as convex envelops of original problem P [5].As the initial step, this approximated problem solves by using branch and bound method to get optimal solution. If the algorithm does not result a global solution in this step, new set of problems are creating for yield the optimal solution. If the original problem is nonconvex it is hard to get optimal solution in this step. Nevertheless, if the problem is convex definitely it will yield the global solution.

3.3. The Approximating Problem

The approximating problem can be obtained by replacing each nonlinear function F_{ij} with a piecewise linear approximation over the interval $[l_j, L_j]$. Then the grid point selection can be define by selecting $p_j + 1$ grid points $y_{j0}, \dots, y_{jp_j}$ in $[l_j, L_j]$ .where $y_{j0} = l_j$ and $y_{jp_j} = L_j$ and using convex combinations of the numbers, $F_{ij}(x_j)$ can be approximated as $F_{ij}(y_{jk})$ and $F_{ij}(y_{j,k+1})$ over the subinterval $[y_{jk}, y_{j,k+1}]$.

Mathematically, we obtain this approximation by setting $F_{ij}(x_j) \cong f_{ij}(\theta_j)$ where

$$f_{ij}(\theta_j) = \sum_{k\epsilon K_j} \theta_{jk} F_{ij}(y_{ij}) \tag{3.1}$$

Where $K_j = \{1,2, \dots, p_j\}, \theta_j = \left(\theta_{j0}, \dots, \theta_{jp_j}\right)$ if

$$\sum_{k\epsilon K_j} \theta_{jk} F_{ij}(y_{ij}) = x_j \tag{3.2}$$

$$\sum_{k\epsilon K_j} \theta_{jk} = 1 \tag{3.3}$$

$$\theta_{jk} \geq 0 \qquad k\epsilon K_j \tag{3.4}$$

This approximated problem is a linear problem with the following additional constraint. i.e.

- at most two of the weights $\{\theta_{jk} : k\epsilon K_j\}$ are nonzero,
- if two are nonzero, then these must correspond to adjacent grid points.

This the definition of separable programming to solve nonlinear programming problem. This last adjacent restriction is needed to consider the problem as a linear

problem. Otherwise, it may yield any point in the convex function, which is found on the vertical line that passes through x_j.

3.3.1 The Adjacent Weights Restriction (AWR)

Let $k \in K_j$be a set of consecutive integers. The set of numbers $\{\theta_{jk}: k\epsilon K_j\}$ satisfies the adjacent weights restriction if at most two of these numbers are nonzero, and if $\theta_{js}, \theta_{jt} > 0$ then either

$s = t - 1$ or $s = t + 1$.

We shall use the symbol $f_{ij}(\theta_j)$ to dente the function defined by expression (3.1) and the symbol to denote the function $f_{ij}(\theta_j)$ constrained by relations (3.2), (3.3), (3.4) and the AWR. Thus $f_{ij}(\theta_j)$ denotes a linear function of the variables $(\theta_{j0}, \theta_{j1} \dots, \theta_{jp_j})$ while $f_{ij}(x_j)$ denotes a piecewise linear function of the single variable x_j. Likewise

$$f_i(\theta) = \sum_{j=1}^{n} f_{ij}(\theta_j)$$

and

$$f_i(x) = \sum_{j=1}^{n} f_{ij}(x_j)$$

for $i = 0,1,2, \dots\dots, m$.

By replacing each $F_{ij}(x_j)$ by its piecewise linear approximation $f_{ij}(x_j)$, following approximate problem can be obtained.

Problem P[1]

$$\begin{aligned}
&minimize && f_0(\theta) = \sum_{j=1}^{n} \sum_{k\epsilon K_j} \theta_{jk} F_{0j} y_{ik} \\
&subject\ to && f_i(\theta) = \sum_{j=1}^{n} \sum_{k\epsilon K_j} \theta_{jk} F_{ij} y_{ik} \ \leq\ b_i \quad i = 1,2 \dots m \\
& && \textstyle\sum_{k\epsilon K_j} \theta_{jk} = 1 \qquad\qquad j = 1,2 \dots n \\
& && \theta_{jk} \geq 0 \qquad k\epsilon K_j \qquad j = 1,2 \dots n \ ;\ k\epsilon K_j \\
& && \{\theta_{jk}: k\epsilon K_j\} \quad satisfies\ AWR\ j = 1,2, \dots\dots, n
\end{aligned}$$

Here $\theta = (\theta_1; \theta_2 \dots, ; \theta_n) = \left(\theta_{10}, \theta_{11} \dots, \theta_{1p_1}; \theta_{20}, \dots, ; \dots; \theta_{n0}, \dots, \theta_{np_n}\right)$.

The approximated solution for original problem P can be obtained by solving this approximated problem P^1. Following relationship can be used to yield the approximated solution point of problem P

$$x_j^* = \sum_{k \epsilon K_j} \theta_{jk}^* y_{jk} \qquad j = 1, \dots, n. \tag{3.5}$$

All other advanced algorithmic steps are available on the Appendix A, which discuss deeper in to the advanced methods with details. Hereafter, this section only forces into main concepts and methodology of the algorithm that are important when developing the package.

3.4. The Branch and Bound Approach

As the first step of the branch and bound algorithm, the approximated problem of the original problem P is solved using a linear programming method (ex: simplex method). In this stage $f_0(\theta)$ and ∞ are set as the initial best lower bound (BLB) and best upper bound (BUB) respectively. The feasibility of the solution is checked a test which, if the solution is feasible, solution is considered as the global solution for problem P. if the solution is not feasible, subset of three or two linear problems are created. The smallest value among the $f_0(\theta)$ value of all the subproblems, is selected as the new best lower bound (BLB). The subproblem corresponds to the smallest lower bound is used to create another three or two linear sub problems.

The corresponding upper bound (UB) and lower bound (LB) are calculated in the following manner using the initial solution of a given problem t.

Assume that now we are in the problem t and compute the x_j^t using 3.5 relationship. Then calculate the $\widetilde{\theta_{jk}^t}$ using following relationship and AWR.

$$x_j^t = \sum_{k \epsilon K_j} \widetilde{\theta_{jk}^t} y_{jk} \qquad j = 1, \dots, n. \tag{3.6}$$

Hence, lower bound and upper bound for a given problem t

$$LB(t) = \begin{cases} f_0(\theta^t) \\ \\ +\infty \end{cases} \qquad UB(t) = \begin{cases} f_0(\tilde{\theta}^t) \\ \\ +\infty \end{cases}$$

Then best lower bound and best upper bound of a given stage are calculated as follows. The set of all intermediate problems at stage l is denoted by $I(l)$. At stage one, $I(1) = \{1\}$, and, if three new problems are created to form stage two, $I(2) = \{2,3,4\}$,

Say we are in stage l,

$$BLB(l) = \min\big(LB(t)\big)\, which\ t\ \in I(l)$$

$$BUB(l) = \min\big(UB(t)\big)\, which\ t = 1,2 \dots \dots, L$$

CHAPTER 04
MATERIALS AND METHODOLOGY

4.1 Overview

This section provides the information about all the steps for **laptimize** python package development. Also, material and methods that are used to develop and publish the package.

4.2 Introduction

Package "**laptimize**" was built on with python 3. Also, linear programming package (Pulp), which is available in the python, was incorporated as the main root package. After implementing the final algorithm laptimize was published in PyPi repository and can be install via `pip install laptimize`.

According to the Flank and Soland [5] [2] [4] [7], the optimization problem should be in following format before applying this algorithm.

$$
\mathrm{P}\left\{\begin{array}{lll}
Minimize: & F_0(x) & \\
Where & x = (x_1, x_2, \dots\dots\dots, x_n) & \\
Subject\ to: & F_i(x) \leq b_i & i = 1,2, \dots\dots\dots, q \\
 & F_i(x) = b_i & i = q+1, \dots\dots\dots, m \\
 & l_i \leq x_i \leq v_i &
\end{array}\right.
$$

We assume here $F_0(x), F_i(x)$ functions are separable and continues and over the bounds of l_i and v_i. i.e., each $F_i(x)$ can be written as

$$F_i(x) = \sum_{j=1}^{n} F_{ij} x_j$$

After converting original problem to format of problem P, problem P^1 will be constructed by laptimize package as following manner. Problem P^1 is the linear approximated from of problem P. Let's consider an original form of $F_i(x)$ within the interval of $l_i \leq x_i \leq v_i$, is shown in the Figure 4.1

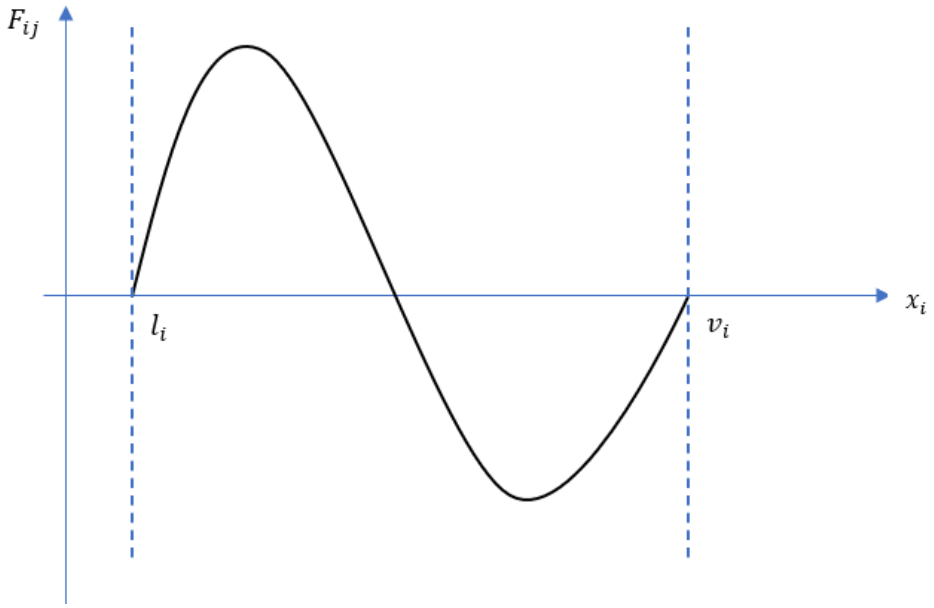


Figure 4.1: Initial $F_i(x)$ representation

As the next step, $[l_i, v_i]$ will be divided in to t intervals by considering the grid points $[y_{i0}, y_{i2}, \dots, y_{ik}, .. y_{it}]$ and where $l_i = y_{i0} \leq y_{i1} \dots . \leq y_{ik} \dots \leq y_{it} = v_i$. As shown is Figure – 2 (t=7), problem Q can be approximated by piecewise linear segments using following function.

Let $y_i = \theta y_{ik} + (1-\theta) y_{i(k+1)}$ for some $\lambda \in [0,1]$, Then,

$$\tilde{F}_{ij}(y_i) = \theta F_{ij}(y_{ik}) + (1-\theta) F_{ij}\left(y_{i(k+1)}\right)$$

The grid points may or may not be equal distance and also accuracy of the approximated function improves as the number of grid point increase

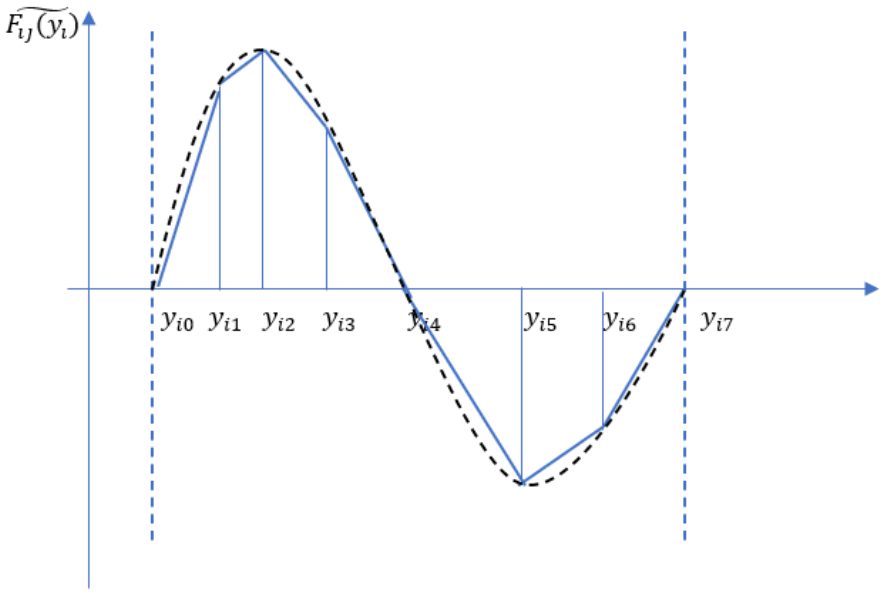


Figure 4.2: The Approximating Function $\widetilde{F_{ij}(y_i)}$ representation

Hereafter laptimize solve the problem according to theory and martials by Flank (1972) and gives the optimal approximated solution. Chapter 3 and the Appendix-A can be referred for more information. The notations that are using hereafter with refer to Appendix A.

$$
\mathrm{P}^1 \begin{cases}
minimize & f_0(\theta) = \sum_{j=1}^{n} \sum_{k \epsilon K_j} \theta_{jk} F_{0j} y_{ik} & & \\
subject\ to & f_i(\theta) = \sum_{j=1}^{n} \sum_{k \epsilon K_j} \theta_{jk} F_{ij} y_{ik} \leq b_i & & i = 1,2 \dots m \\
& \sum_{k \epsilon K_j} \theta_{jk} = 1 & & j = 1,2 \dots n \\
& \theta_{jk} \geq 0 & k \epsilon K_j & j = 1,2 \dots n \ ; \ k \epsilon K_j \\
& \{\theta_{jk} : k \epsilon K_j\} & satisfies\ AWR & j = 1,2, \dots . , n
\end{cases}
$$

4.2.1 Python

Python is the popular programming language in recent decades for web development to data science. It has a very mature and supportive development community with hundreds of ready-to-use packages for users. These ready-to-use packagers allow us to reduce our time in the initial development cycle. One can use it as a scripting language as well as a programming language. Therefore, python was selected as the programming language to develop the laptimize.

4.2.2 PyPi Repository

Simply it is GitHub for python packages. **PyPI (Python Package Index)** is a software repository for the Python programming language. The PyPI repository aids users in locating and installing Python community-developed and shared applications. One can publish an own python package by creating an account in PyPi. In addition to the PyPi repository test PyPi is available for testing purposes of development community.

4.2.3 Pulp package in python

Pulp is the main base package and dependency of laptimize. It is an open-source linear programming (LP) package that uses a variety of standard LP solvers such as 'GLPK_CMD', 'CPLEX_CMD', 'GUROBI_CMD', 'PULP_CBC_CMD' etc. Linear Programming and Integer Programming models can be express as similar to conventional mathematical notation. 'PULP_CBC_CMD' was used as the default solver in the laptimize for solve the linear subproblems.

4.2.4 Mystic

Mystic was used as a python package to solve selected problems and compare results against the laptimize. This is a python package that provides a collection of Optimization algorithms newly developed by Mike Mckerns.

4.2.5 Scipy.optimize

Scipy.optimize also enrich with lot of option for solving the objective functions with possibly subject to constraints. The main limitation of scipy.optimize is a dependency for the initial values of the variables. It enriches with solver for linear programming, nonlinear programming, root finding and curve fitting.

4.2.4 Creating an Own Python Package

After developing a publish-ready laptimize code, PyPi repository guidelines were followed to packaging the python project. The __init__.py file was added to the package to make Python interpret the laptimize package directory as a Python package. Other required files such as setup.py, readme.md, and LICENSE.txt added as described in PyPi guidelines.

4.3 laptimize –Linear Approximated Programming for Optimization.

4.3.1 Project Description

Four main python classes were developed to build the laptimize package. Source code is available under the GitHub link https://github.com/uichathurika/laptimize.

Public listing in PyPi repository can be found in the link: https://pypi.org/project/laptimize/

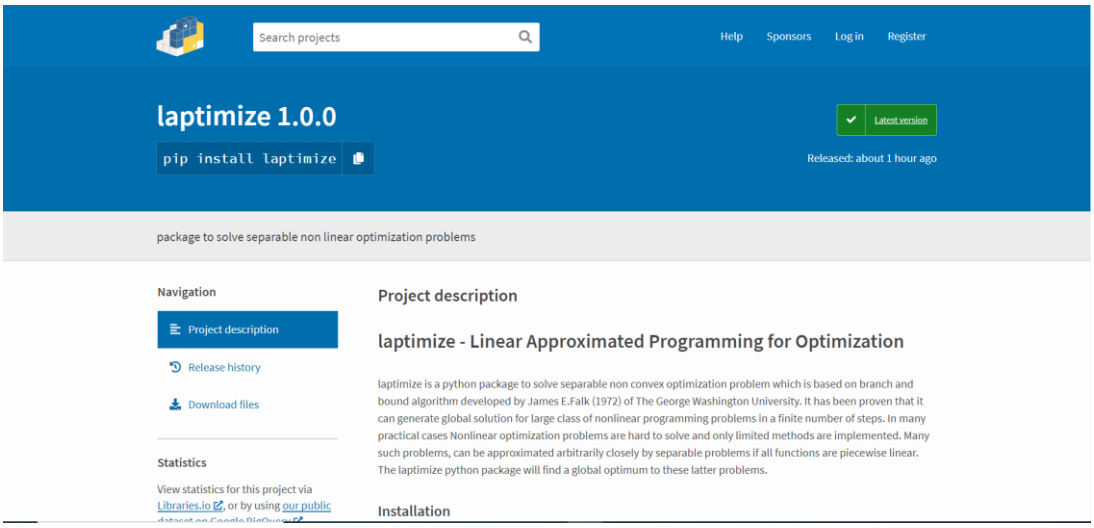


Figure 4.3: laptimize package page in PyPi

laptimize can install via pip

Figure 4.4: laptimize installation via pip

Also, multiple types of nonlinear separable examples were available under the '**examples'** directory.

When building packages in python, it is required to add an `__init__.py` file in every sub directory in the package. As laptimize have only one main directory `__init__.py` was add inside the laptimize directory. Each and every method is complete with the following doc-strings

- **Summary:** Simple description about the process that function supposed to do
- **Parameters:** Explanation about the parameters and parameter-types which the function expects.
- **Return value:** Function return values and their types.

There for users can get better understanding about each method by referring the code itself. The project structure of the laptimize can be illustrate as Figure 4.6.

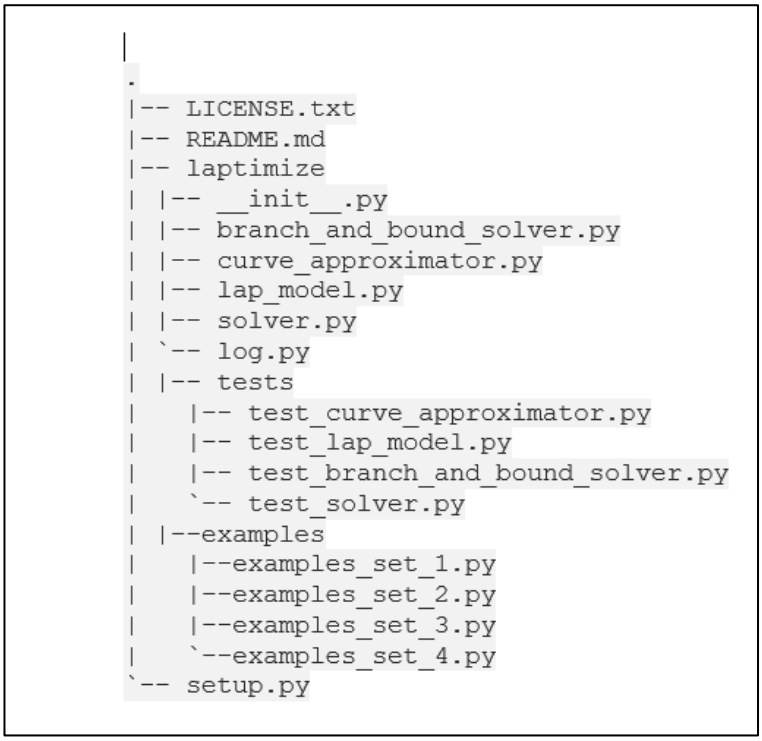


Figure 4.6: laptimize project structure

`LICENSE.txt`

The main objective of the study was developed a package which is publicly available, therefore MIT license were used.

`Setup.py`

The `setup.py` file contains information about the package, specifically the *name* of the package, its *version,* platform-dependencies and a whole lot more. Also, this includes all the information python installer needs to know when installing the package.

```
from setuptools import setup, find_packages

try:
    import pypandoc

    long_description = pypandoc.convert('README.md', 'rst')
except(IOError, ImportError):
    long_description = open('README.md').read()

VERSION = '1.0.0'
DESCRIPTION = 'package to solve separable nonlinear optimization problems'
URL = 'https://github.com/uichathurika/laptimize'
INSTALL_REQUIRES = [
    'numpy>=1.19.2',
    'pandas>=1.1.3',
    'pulp>=2.3'

]
LICENSE = 'MIT'

setup(
    name="laptimize",
    version=VERSION,
    author="Ishanga Udatiyawala",
    author_email="uichathurika@gmail.com",
    description=DESCRIPTION,
    long_description=long_description,
    long_description_content_type="text/markdown",
    packages=find_packages(),
    install_requires=INSTALL_REQUIRES,

    keywords=['python', 'nonlinear optimization', 'separable programming',
'branch and bound', 'global solution'],
    classifiers=[
        "Development Status :: 3 - Alpha",
        "Intended Audience :: Education",
        "Programming Language :: Python :: 3.6",
        "Operating System :: Microsoft :: Windows",
        "Natural Language :: English",
        "Topic :: Scientific/Engineering :: Mathematics"
    ]
)
```

Figure 4.7: laptimize setup.py file

4.3.2 User's Guide - laptimize

Followings are main steps that user need to follow when using laptimize.

Consider the following example [5];

$$Minimize: \quad f_0(x_1, x_2) = 2x_1^3 - 9\,x_1 + 9x_1^2 - 2x_2^3 + 9x_2^2 - 9x_2$$

$$Subject\ to: \quad f_1(x_1, x_2) = -6x_1^2 + 18x_2 \leq 9$$

$$f_2(x_1, x_2) = 6x_1^2 - 18x_2 \leq 0$$

$$0 \leq x_{1,} x_2 \leq 3$$

Problem Related Variables

- Number of variables: for this example, it is two
- Number of constraints: inequalities under the subject to section except variable bounds
- Variable bounds: minimum and maximum value that a given variable can take

After identifying the above components problem can be famulated as follows. All functions should be input in lambda format.

```
example = {
    'objective': {'x1': lambda x: 2 * x ** 3 - 9 * x ** 2 + 9 * x,
'x2': lambda x: -2 * x ** 3 + 9 * x ** 2 - 9 * x},
    'constraints_1': {'x1': lambda x: -6 * x ** 2, 'x2': lambda x:
18 * x, 'value': 9},
    'constraints_2': {'x1': lambda x: 6 * x ** 2, 'x2': lambda x: -
18 * x, 'value': 0},
    'capacity': {'x1': [0, 3], 'x2': [0, 3]}}
```

Main Input Parameters

- Constraints: python dictionary, problem definition which formulated as
- Partition_len: distance between the two grid points, if the variable bound gap is less than one, it is better to use small partition length such as 0.025. Also, with large variable bound gap 1 or 0.5 is more effective than default partition length 0.25. Small partition lengths will be increased run time when bound gap is large.
- error: error input value is used as a tolerance in many stages inside the algorithm.

  When BUB – BLB < error, the problem will be considered as solved

When $\theta_k + \theta_{k+1} > 1$ – error, considered as AWR is satisfied,

Default value is 0.001for error.

After setting the above parameters above example can be solved using following python implementation.

```
example_1 = {
    'objective': {'x1': lambda x: 2 * x ** 3 - 9 * x ** 2 + 9 * x, 'x2':
lambda x: -2 * x ** 3 + 9 * x ** 2 - 9 * x},
    'constraints_1': {'x1': lambda x: -6 * x ** 2, 'x2': lambda x: 18 * x,
'value': 9},
    'constraints_2': {'x1': lambda x: 6 * x ** 2, 'x2': lambda x: -18 * x,
'value': 0},
    'capacity': {'x1': [0, 3], 'x2': [0, 3]}}

solution_1 = Solver(example_1, partition_len=0.25).solve()
print(solution_1[0])
print(solution_1[1])
```

Expected laptimize output

- The first position of the solution includes the optimal solution for each variable and corresponding objective value.

  ```
  {'x1': 1.75, 'x2': 1.02083333625, 'obj_value': -3.0182}
  ```

- The second position of the solution consists of a python pandas data frame which includes all the local solutions and global solution.

In additional to the main output laptimize also collects following intermediate parameters

Intermediate Parameters

- Lower bound: lower bound value in each solving stage
- Upper bound: upper bound value in each solving stage
- Segment values: segment values for each variable
- Curve values: function values for objective and constraints for each segment
- K, K^+, K^-: K, K upper and K lower values are generated in each solution stage
- Branching variable: branching variable in each solving stage, if any

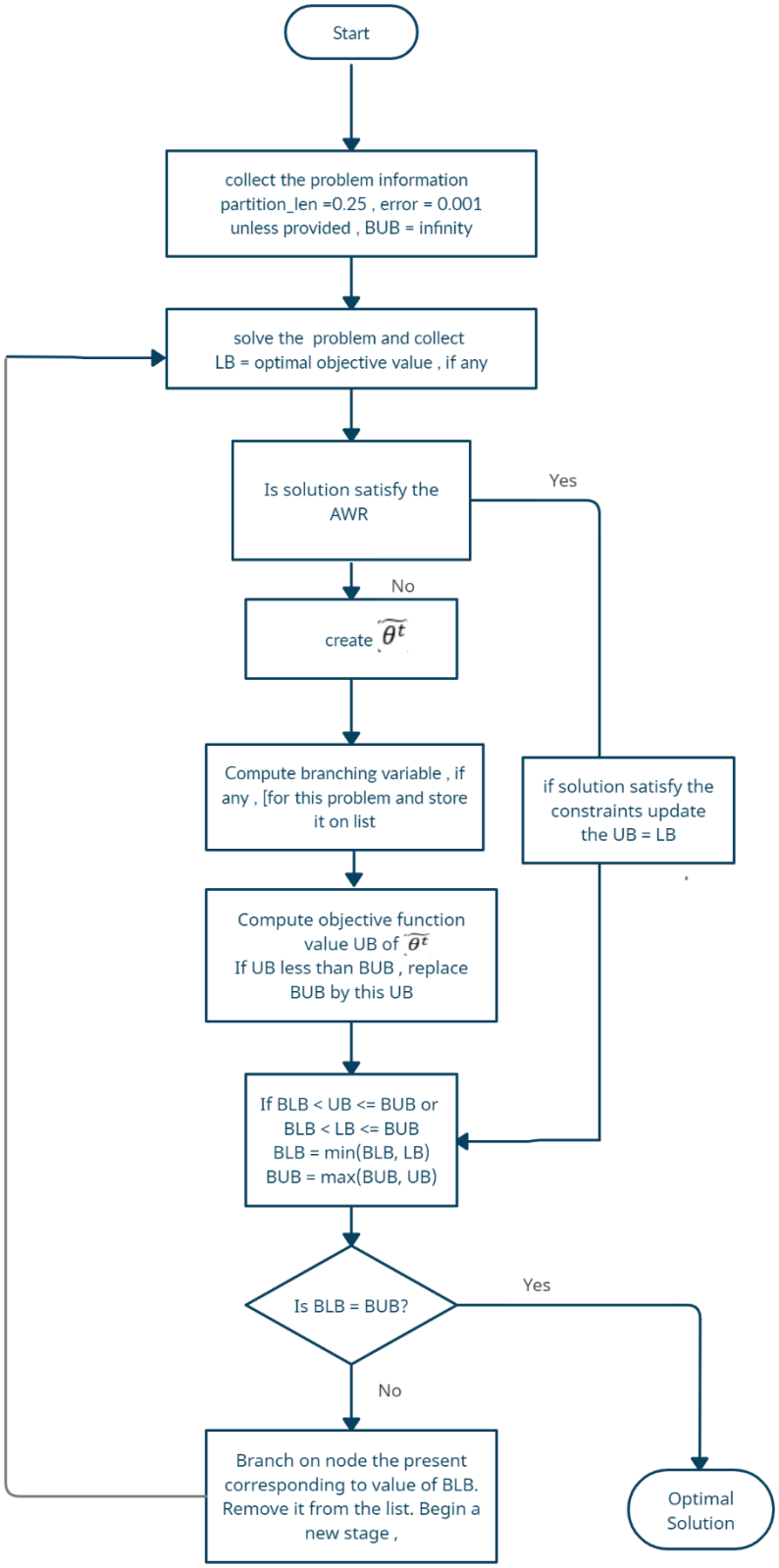


Figure 4.8: laptimize algorithm flow chart

CHAPTER 05
RESULTS AND DUSCUSSION

5.1 Overview

In this chapter, the results obtained through lapimize package were discussed and compared. The main objective of this research was to build a publicly available python package to solve separable nonconvex problems. The correctness of the implementation and validation of the results was done by comparing the solution of selected nonlinear (convex and concave) optimization problems. Run time and the correctness of the solutions were measured for 10 nonlinear optimization problems comparing results against the scipy.optimize and mystic packagers. Examples were selected from the literature papers which have known optimal global solutions.

Results were obtained using the default tuning parameters such as iterations, tolerance, and variable initialization. Problems related to input parameters such as variable bounds, objective type (minimize/maximize), and constrains equalities are common across the packagers. Lower bound of the variables was used as the initial variable values for the scipy.optimize and mystic since it is required as a positional argument.

Mystic 0.3.7, Scipy 1.5.4 were used to compare results agents to the initial version of laptimize 1.0.0 with python 3.7.2. The computations were performed on an Intel(R) Core(TM) i5-250U processor 1.60Gz/16GB PC compatible. Each type of problem was run in 3 times.

All the examples that are discussed in this section are included in the laptimize package under the Examples directory with all the information. It will be a benefit for the laptimize users to get a better understanding of the use cases the applicability of the package. Further, all the examples were converted to the format of problem P as in chapter 4 before feeding to the laptimize.

5.2 Results

5.2.1 Example 01

Source: [5]

$$Minimize: \quad : f_0(x_1, x_2) = f_0(x_1) + f_0(x_2)$$

$$where: \quad f_0(x_1) = 2x_1^3 - 9x_1 + 9x_1^2$$

$$f_0(x_2) = -2x_2^3 + 9x_2^2 - 9x_2$$

$$Subject\ to: \quad f_1(x_1, x_2) = -6x_1^2 + 18x_2 \leq 9$$

$$f_2(x_1, x_2) = 6x_1^2 - 18x_2 \leq 0$$

$$0 \leq x_1, x_2 \leq 3$$

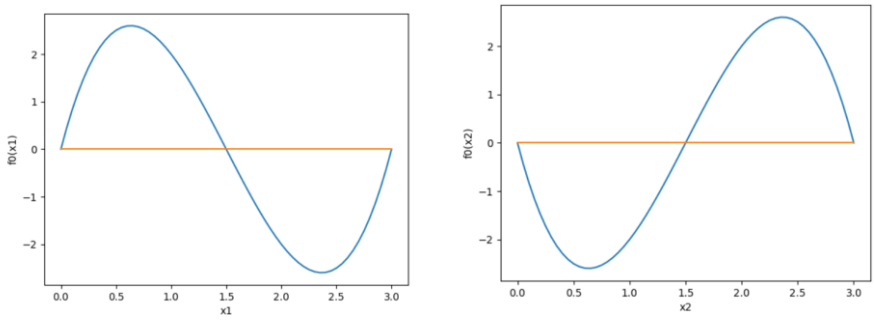


Figure 5.1: Graphical representation of objective function

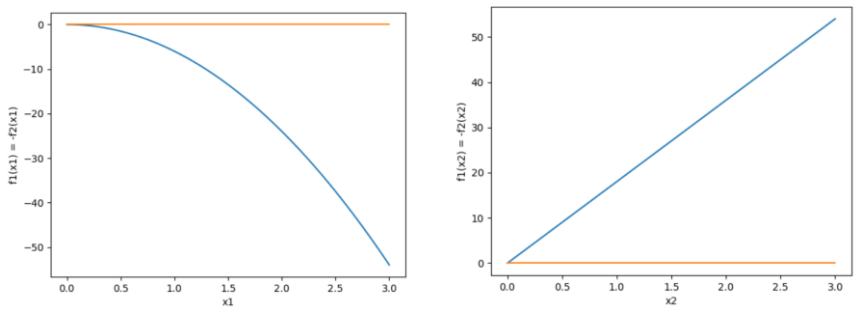


Figure 5.1: Graphical representation of nonlinear constraint function

Some intermediate results are represented in the following tables. All the notations are related to the algorithm in chapter 3 and appendix A.

Table 5.1 Approximated function values for grid points

$variable\ key$	$grid\ point\ key$	$grid\ points$	$f_0(y_{ik})$	$f_1(y_{ik})$	$f_2(y_{ik})$
x_1	$y_{1,0}$	0	0	0	0
x_1	$y_{1,1}$	0.25	1.71875	-0.375	0.375
x_1	$y_{1,2}$	0.5	2.5	-1.5	1.5
x_1	$y_{1,3}$	0.75	2.53125	-3.375	3.375
x_1	$y_{1,4}$	1	2	-6	6
x_1	$y_{1,5}$	1.25	1.09375	-9.375	9.375
x_1	$y_{1,6}$	1.5	0	-13.5	13.5
x_1	$y_{1,7}$	1.75	-1.09375	-18.375	18.375
x_1	$y_{1,8}$	2	-2	-24	24
x_1	$y_{1,9}$	2.25	-2.53125	-30.375	30.375
x_1	$y_{1,10}$	2.5	-2.5	-37.5	37.5
x_1	$y_{1,11}$	2.75	-1.71875	-45.375	45.375
x_1	$y_{1,12}$	3	0	-54	54
x_2	$y_{2,0}$	0	0	0	0
x_2	$y_{2,1}$	0.25	-1.71875	4.5	-4.5
x_2	$y_{2,2}$	0.5	-2.5	9	-9
x_2	$y_{2,3}$	0.75	-2.53125	13.5	-13.5
x_2	$y_{2,4}$	1	-2	18	-18
x_2	$y_{2,5}$	1.25	-1.09375	22.5	-22.5
x_2	$y_{2,6}$	1.5	0	27	-27
x_2	$y_{2,7}$	1.75	1.09375	31.5	-31.5
x_2	$y_{2,8}$	2	2	36	-36
x_2	$y_{2,9}$	2.25	2.53125	40.5	-40.5
x_2	$y_{2,10}$	2.5	2.5	45	-45
x_2	$y_{2,11}$	2.75	1.71875	49.5	-49.5
x_2	$y_{2,12}$	3	0	54	-54

The given global solution in the paper for approximated problem is [1.714,1.000] with the objective value of -2.857. According to the laptimize there are local solutions near the points (0,0.5), (2.11, l.50) and (2 .74, 3.00) with values -2.50, -2.25 and -1.75 respectively. Also approximated global solution is found to be the point [1.75, 1.02] with -3.0182 objective function value, which is more accurate than the approximated solution given in the paper.

As the first step of the algorithm P [1]version of example 01 is solved and obtained initial solution details are as follows

Table 5.2.1: Initial solution for Example 01

Variables	Index state	Solution θ_1^t θ_2^t	$LB(\theta^t)$ $f_0(\theta^t)$
x_1	[0,1,2,3,4,5,7,8,9,10,11,12]	[0,0,0,0,0,0,0,0,0,1,0,0,0]	-4.0013
x_2	[0,1,2,3,4,5,7,8,9,10,11,12]	[0,0,0,0.583,0,0,0,0,0,0,0,0,0.416]	

According to the above table results, initial solution θ^tdoes not satisfy the **Adjacent Weights Restriction** rule. Therefore $\widetilde{\theta^t}$ values have been calculated and initial lower bound set to $LB(\theta^t)$ = -4.0013 and upper bound set to $LB(\theta^t)$ = -1.7104.

Table 5.1.2: Initial solution for Example 01

Solution $\widetilde{\theta_1^t}$ $\widetilde{\theta_2^t}$	$UB(\theta^t)$ $f_0(\widetilde{\theta^t})$	Branching variable	Best bounds $BLB(\theta^t)$ $BUB(\theta^t)$
[0,0,0,0,0,0,0,0,0,1,0,0,0]	-1.7104	x_2	-4.0013
[0,0,0,0,0,0,0.25,0.75,0,0,0,0,0]			-1.7104

Considering the branching rule, three sub-problems is created from the initial problem. This step is named the second stage of the algorithm. According to Flank [5] if this step is selected for further branching then at most three new subproblems are created.

Figure 5.2 illustrate the branch and bound tree of this problem. As intermediate steps eight subproblems were created in four stages. Node N1 was created in the first stage, after that nods N2, N3, and N4 are created in the second stage. Likewise, nodes N5 and N6 are formed in the third stage. As final stage, nodes N7 and N8 are created and global solution is found in node 8.

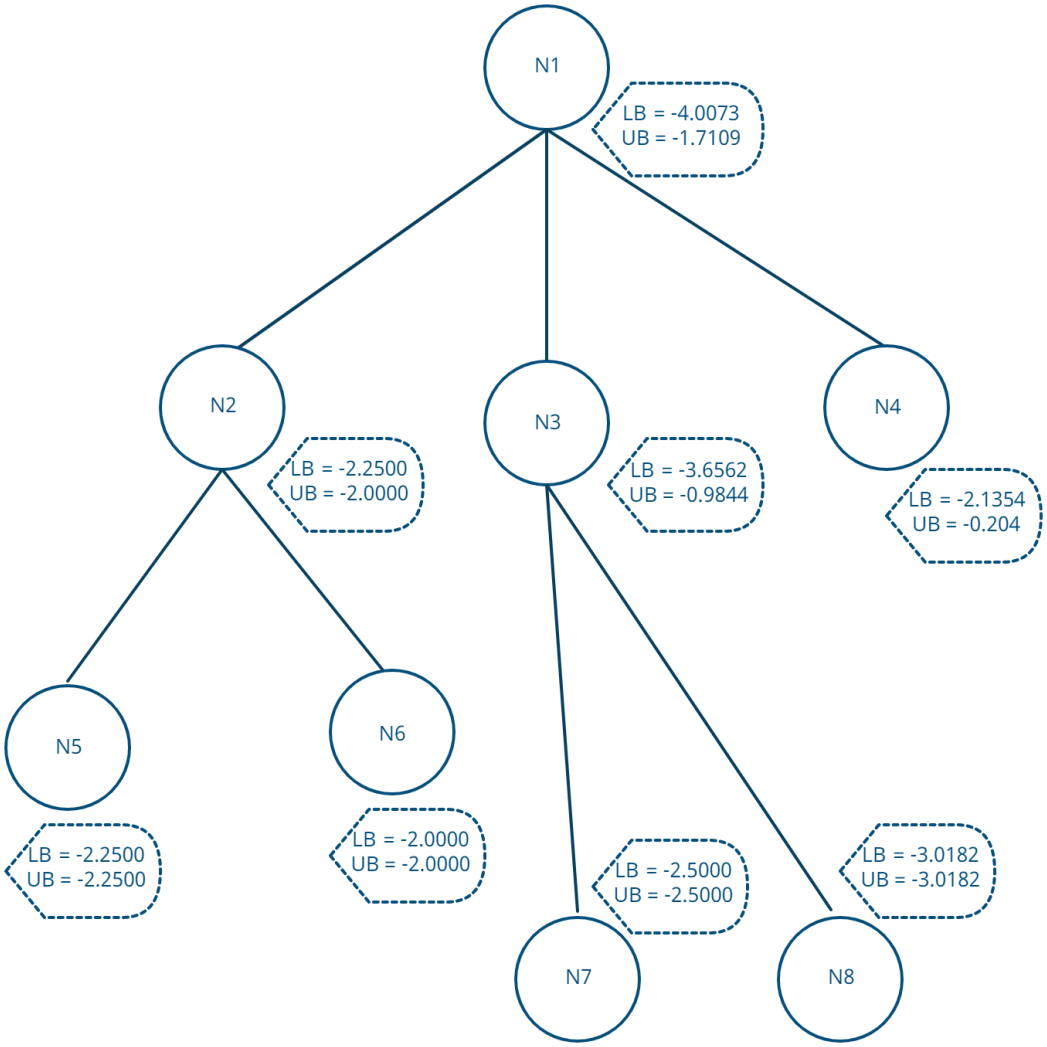


Figure 5.3: Branch and Bound tree for Example 01

Table 5.3: Results Comparison for Example 01

Package	Objective Value	Final Solution	Time(s)
Laptimize	-3.0182	x_1: 1.75, x_2: 1.02	1.0261
scipy.optimize	-1.9790	x_1:1.65, x_2: 1.19	0.0169
Mystic	-2.9237	x_1:1.65, x_2: 0.91	0.03472

According to the table 5.3 laptimize gives the most accurate solution with the default tuning parameters while mystic also performed well to approximate the solution.

5.2.2 Example 02

Source: [4]

$Minimize$: $f_0(x_1, x_2) = 12\,x_1 - x_2^2 + 7x_2$

$Subject\ to$:

$convex\ polygon$ $f_1(x_1, x_2) = 2x_1 + x_2 \leq 3$

$non\ convex\ constraint$ $f_1(x_1, x_2) = -2x_1^4 - x_2 \leq -2$

$$0 \leq x_1 \leq 2$$

$$0 \leq x_2 \leq 3$$

This example consists of concave objective function, one concave constraint and one linear constraint. Reference to the paper, point [0.00, 2.00] solves the problem with 10.00 objective value. According to the laptimize solution, there are two local solutions near the points [1.00, 0.00] and [0.5, 1.75] with objective values 12.00 and 15.56.

Table 5.4: Results Comparison for Example 02

Package	Objective Value	Final Solution	Time(s)
Laptimize	10.00	x_1: 0.00, x_2: 2.00	0.6988
scipy.optimize	16.59	x_1: 0.63, x_2: 1.71	0.0101
Mystic	12.00	x_1: 1.00, x_2: 0.00	1.3001

As seen in table 5.4 laptimize have reach the expected global optimum, while mystic ends up with the local optimum.

5.2.3 Example 03

Source: [2]

$Minimize$: $f_0(x_1, x_2) = f_0(x_1) + f_0(x_2)$

$Where$;

$$f_0(x_1) = 0 \qquad x_1 = 0$$

$$= -x_1^2 + 4x_1 + 2 \qquad 0 < x_1 \leq 6$$

$$f_0(x_2) = 2 \qquad x_2 = 0$$

$$= -x_2^2 + 2x_1 + 4 \qquad 0 < x_2 \leq 5$$

$Subject\ to$: $f_1(x_1, x_2) = -x_1 + 3x_2 \leq 5$

$$f_1(x_1, x_2) = 2x_1 - x_2 \leq 5$$

$$f_1(x_1, x_2) = -2x_1 + x_2 \leq 0$$

$$f_1(x_1, x_2) = x_1 - 3x_2 \leq 0$$

$$0 \leq x_1 \leq 6, 0 \leq x_2 \leq 5$$

Example 03 consists of concave objective function which is discontinuity at zero. Constraints indicate a convex polygon. Paper indicates that global solution occurs at origin with objective value 2.00.

Table 5.5: Results Comparison for Example 03

Package	Objective Value	Final Solution	Time(s)
Laptimize	2.00	x_1: 0.00, x_2: 0.00	1.6043
scipy.optimize	2.00	x_1: 0.00, x_2: 0.00	0.0156
Mystic	2.00	x_1: 0.00, x_2: 0.00	0.0377

All three packagers have got the correct solution for this problem. laptimize intimidate results indicate the there is a local solution near the point [4,3] with objective value 3.00.

5.2.4 Example 04

Source: [16]

$Minimize$: $f_0(x_1, x_2) = f_0(x_1) + f_0(x_2)$

$Where$:

$$f_0(x_1) = 5\,x_1 - x_1^2$$

$$f_0(x_2) = -x_2^2 + 3x_2$$

$Subject\ to$: $f_1(x_1, x_2) = 2x_1^4 + x_2 \leq 32$

$$f_2(x_1, x_2) = x_1 - 2x_2^2 \leq 32$$

$$0 \leq x_1 \leq 2, 0 \leq x_2 \leq 4$$

Table 5.6: Results Comparison for Example 04

Package	Objective Value	Final Solution	Time(s)
Laptimize	-8.2255	x_1: 1.97, x_2: 1.50	0.2702
scipy.optimize	-2.6189	x_1: 2.00, x_2: 3.87	0.0080
Mystic	-8.0032	x_1: 1.82, x_2: 1.31	0.0313

This is a convex example which separable programming gives global solution in the first step. Above comparison indicates that laptimize gives the lowest objective value which satisfy the constrains.

5.2.5 Example 05

Source: [2]

$Minimize$: $f_0(x_1, x_2) = f_0(x_1) + f_0(x_2)$

$Where$:

$$f_0(x_1) = -3(x_1 - 0.5)^2 \qquad 0 \leq x_1 \leq 0.5$$

$$= 3(x_1 - 0.5)^2 \qquad 0.5 \leq x_1 \leq 1$$

$$f_0(x_2) = -2x_2^2 \qquad 0 \leq x_2 \leq 2$$

$Subject\ to$: $f_1(x_1, x_2) = -x_1 + 2x_2 \leq 1$

$$0 \leq x_1 \leq 1, 0 \leq x_2 \leq 2$$

As mention in the paper $f_0(x_2)$ is concave, but $f_0(x_1)$ is concave in the interval [0, 0.5] and convex in the interval [0.5, 1] and optimum point is locating near the [0.8, 0.9] with objective value 1.35.

Table 5.7: Results Comparison for Example 05

Package	Objective Value	Final Solution	Time(s)
Laptimize	-1.375	x_1: 0.75, x_2: 0.875	0.3477
scipy.optimize	-8.7499	x_1: 0.00, x_2: 2.00	0.0080
Mystic	-0.9659	x_1: 0.26, x_2: 0.63	0.0184

According to the Table 5.7 all three packagers failed to generate the global solution with default parameters. Therefore 0.025 partition length was used get optimal solution from the laptimize. It will indicate that smaller partition length is more suitable for problems which have small variable bounds (one or less than one).

5.2.6 Example 06

Source: [1]

$Minimize$: $f_0(x_1, x_2, x_3) = f_0(x_1) + f_0(x_2) + f_0(x_3)$

$Where$:

$$f_0(x_1) = -6\,x_1 + x_1^2$$
$$f_0(x_2) = x_2^2 - 8x_2$$
$$f_0(x_3) = 0.5x_3$$

$Subject\ to$: $f_1(x_1, x_2, x_3) = x_1 + x_2 + x_3 \leq 5$

$$f_2(x_1, x_2) = x_1^2 - x_2 \leq 3$$
$$0 \leq x_1, x_2, x_3 \leq 5$$

Table 5.8: Results Comparison for Example 06

Package	Objective Value	Final Solution	Time(s)
Laptimize	-23.00	x_1: 2.00, x_2: 3.00, x_3: 0.00	0.1067
scipy.optimize	-27.49	x_1: 3.00, x_2: 4.00, x_3: 0.00	0.0049
Mystic	-19.68	x_1: 0.94, x_2: 2.65, x_3: 5.00	0.0295

This problem is an example for a convex separable nonlinear optimization problem. Therefore, an optimal solution to the approximating linear programming problem is a feasible solution to the original problem in first step itself. No feather branching is need to solve this problem. laptimize has obtained the global solution in the first stage.

5.2.7 Example 07

Source: [17]

$Minimize$: $f_0(x_1, x_2, x_3, x_4) = f_0(x_1) + f_0(x_2) + f_0(x_3) + f_0(x_4)$

$Where$:

$$f_0(x_1) = 1.5\,x_1 + 0.5x_1^2$$
$$f_0(x_2) = x_2$$
$$f_0(x_3) = -0.25x_3^2 + 0.75x_3$$
$$f_0(x_4) = x_4 - 1$$

$Subject\ to$: $f_1(x_1, x_2) = x_1^2 - x_2 \leq 0$

$$f_2(x_1, x_2) = (x_1 - 1)^2 - x_2 - 2.5 \leq 0$$
$$f_3(x_3, x_4) = x_3^2 - x_4 \leq 0$$
$$f_4(x_3, x_4) = (x_1 - 1)^2 - x_2 - 2.5 \leq 0$$

Table 5.9: Results Comparison for Example 07

Package	Objective Value	Final Solution	Time(s)
Laptimize	-3.8125	x_1: 0.00, x_2: -0.75, x_3:1.5, x_4: -2.25	32.5132
scipy.optimize	-11.4325	x_1: -1.58, x_2: -2.5, x_3: 1.58, x_4: -2.5	0.0059
Mystic	-4.6458	x_1:0.5, x_2: -2.24, x_3: 0.83, x_4: -2.47	2.9770

Since there are four variables in this problem the running time of the laptimize algorithm was high. But it has converged in to global solution which is given in the reference paper. This indicates that convergence time laptimize algorithm have more dependency with the number of variables in the optimization problem.

5.2.8 Example 08

Source: [18]

$Minimize$: $f_0(x_1, x_2) = f_0(x_1) + f_0(x_2)$

$Where$:

$$f_0(x_1) = -3x_1 + 2x_1^2$$

$$f_0(x_2) = 2x_2$$

$Subject\ to$: $f_1(x_1, x_2) = 3x_1^2 + 4x_2^2 \leq 8$

$$f_2(x_1, x_2) = -3x_1^2 + 12x_1 - 5x_2^2 + 20x_2 \leq 22$$

$$f_3(x_1, x_2) = 3x_1^2 - 12x_1 + 5x_2^2 - 20x_2 \leq -11$$

$$0 \leq x_1 \leq 1.75, 0 \leq x_2 \leq 1.5$$

Table 5.10: Results Comparison for Example 08

Package	Objective Value	Final Solution	Time(s)
Laptimize	-0.8089	x_1: 0.925, x_2: 0.127	0.3826
scipy.optimize	-0.8089	x_1: 0.922, x_2: 0.128	0.0222
Mystic	4.99375	x_1: 0.171, x_2: 0.108	0.0303

Scipy.optimize and laptimize have identified the optimal solution as in the literature. According to the paper this was not formulated as separable function. For implementation purpose problem was converted in to separable function. Laptimize have identified the solution in the first step as this is a convex set optimization problem

5.2.9 Example 09

Source: [9]

$Minimize$: $f_0(x_1, x_2, x_3) = f_0(x_1) + f_0(x_2) + f_0(x_3)$

$Where$:

$$f_0(x_1) = 40x_1 + 5(|x_1 - 10| + x_1 - 10) - 5(|x_1 - 12| + x_1 - 12)$$

$$f_0(x_2) = 20x_1 - 5(|x_2 - 10| + x_2 - 10) + 5(|x_2 - 12| + x_2 - 12)$$

$$f_0(x_1) = 30x_3 + 10(|x_3 - 10| + x_3 - 10) - 5(|x_3 - 20| + x_3 - 20)$$

$Subject\ to$:

$$f_1(x_1, x_2, x_3) = -90x_1 - 20x_2 - 40x_3 \leq -2000$$

$$f_2(x_1, x_2, x_3) = -30x_1 - 80x_2 - 60x_3 \leq -1800$$

$$f_3(x_1, x_2, x_3) = -10x_1 - 20x_2 - 60x_3 \leq -1500$$

$$0 \leq x_1 \leq 12, 0 \leq x_2 \leq 12, \ 0 \leq x_3 \leq 20$$

Table 5.11: Results Comparison for Example 09

Package	Objective Value	Final Solution	Time(s)
Laptimize	1430	x_1: 11.04, x_2: 12.0, x_3:1.16	0.3037
scipy.optimize	0	x_1: 0.00, x_2: 0.00, x_3:0.00	0.0153
Mystic	0	x_1: 0.00, x_2: 0.00, x_3:0.00	1.3664

This problem is already piecewise linear separable problem which can be easily handle with laptimize. According to the literature, objective function is a non-convex cost function and constraints are linear. Optimal solution can be obtained by laptimize with default parameters.

5.2.10 Example 10

Source: [9]

$Minimize$: $f_0(x_1, x_2) = f_0(x_1) + f_0(x_2)$

$Where$:

$f_0(x_1) = 0.23256(x_1 - 11) + 0.00872(|x_1 - 54| + x_1 - 54) - 0.04924(|x_1 - 142| + x_1 - 142)$

$f_0(x_2) = 0.22727(x_1 - 11) + 0.040475(|x_1 - 55| + x_1 - 55) - 0.041865(|x_1 - 201| + x_1 - 201)$

$Subject\ to$: $f_1(x_1, x_2) = x_1 + x_2 = 450$

$0 \le x_1 \le 241, 0 \le x_2 \le 250$

Table 5.12: Results Comparison for Example 10

Package	Objective Value	Final Solution	Time(s)
Laptimize	-106.0	x_1: 200, x_2: 250	0.3653
scipy.optimize	-113.0	x_1: 241, x_2: 250	0.02970
Mystic	5.0385	x_1: 0., x_2: 250	0.0818

This is a real-world example for non-convex objective function. According to the paper, objective function represents the quantity of electricity generated through two hydro-electric generating unit.

The obtained solutions from the laptimize are the same as the ones found in paper. Other two packagers unable to find the optimal solution with default parameters

5.2.11 Example 11

Source: [9]

$Minimize$: $f_0(x_1, x_2) = f_0(x_1) + f_0(x_2)$

$Where$:

$$f_0(x_1) = x_1^3 - 4x_1^2 + 2x_1$$
$$f_0(x_2) = x_2^3 - 4x_2^2 + 3x_2$$

$Subject\ to$: $f_1(x_1, x_2) = 3x_1 + 2x_2 \leq 11.75$

$$f_2(x_1, x_2) = -2x_1 + 5x_2^{1/2} + x_2 \leq -9$$
$$0 \leq x_1 \leq 5, 0 \leq x_2 \leq 4$$

Table 5.13: Results Comparison for Example 11

Package	Objective Value	Final Solution	Time(s)
laptimize	-6.4792	x_1: 2.41, x_2: 2.25	0.0928
scipy.optimize	5. 9755	x_1: 3.87, x_2: 0.07	0.0039
Mystic	6.2195	x_1: 3. 88, x_2: 0.04	0.0585

This is a nonlinear optimization problem with nonconvex objective function and one nonconvex constraint function [9]. According to the literature source global solution resides at [2.3875,2.2155] with -6.5291 objective value. Laptimize have obtained a better global approximated solution with default patition_len parameter. One can obtain more accurate solution by reducing patition_len (ex:0.025)

5.2.12 Example 12

Source: [18]

$$Minimize: \quad f_0(x_1, x_2) = f_0(x_1) + f_0(x_2)$$
$$f_0(x_1) = 3x_1 - 2x_1^2$$
$$f_0(x_2) = -2x_2$$

$$Subject\ to: \quad f_1(x_1, x_2) = 3x_1^2 + 4x_2^2 \leq 8$$
$$f_2(x_1, x_2) = -3x_1^2 + 12x_1 - 5x_2^2 + 20x_2 \leq 22$$
$$f_3(x_1, x_2) = 3x_1^2 - 12x_1 + 5x_2^2 - 20x_2 \leq -11$$
$$0 \leq x_1 \leq 1.75, 0 \leq x_2 \leq 1.5$$

Table 5.14: Results Comparison for Example 12

Package	Objective Value	Final Solution	Time(s)
Laptimize	-2.8283	x_1: 0.000, x_2: 1.4141	0.6997
scipy.optimize	-3.0000	x_1: 0.000, x_2: 1.5	0.0087
Mystic	-1.9647	x_1: 1.75, x_2: 0.5448	2.2977

According to the literature source this is a maximization problem and global maximum resides at [0,1.414]. Problem is converted to minimization problem and solve using three packages. Even if scipy.optimize gives the minimum objective value it does not satisfy the constraints conditions. laptimize solution can be consider as global solution as it is same as the solution given in the literature.

5.2.13 Example 13

Source: [19]

$Minimize$: $f_0(x) = \sin(x) + \sin\left(\frac{10x}{3}\right) + \ln(x) - 0.84x$

$Subject\ to$: $2.7 \leq x \leq 7.5$

Table 5.15: Results Comparison for Example 13

Package	Objective Value	Final Solution	Time(s)
Laptimize	-4.6013	$x = 5.2$	0.0826
scipy.optimize	-4.6013	$x = 5.2$	0.0021
Mystic	-0.5201	$x = 2.719$	0.0045

This is a signal variable example which indicates that laptimize can handle other mathematical expressions such as trigonometric terms and logarithms terms. According to the literature, objective function belongs to convex class. The lower bound of the variable is considered as initialization value for scipy.optimize and mystic. laptimize and scipy.optimize generated the global value while mystic have stick to the initial value.

5.2.14 Example 14

Source: [20]

$$Minimize: \quad f_0(x_1, x_2) = f_0(x_1) + f_0(x_2)$$

$$f_0(x_1) = -10x_1 + 2x_1^2$$

$$f_0(x_2) = -4x_2 + 3x_2^2$$

$$Subject\ to: \quad f_1(x_1, x_2) = 2x_1 + x_2 \leq 5$$

$$0 \leq x_1 \leq 5, 0 \leq x_2 \leq 2.5$$

Table 5.16: Results Comparison for Example 14

Package	Objective Value	Final Solution	Time(s)
Laptimize	-13.6250	x_1: 2.25, x_2: 0.50	0.0748
scipy.optimize	-13.8333	x_1: 2.50, x_2: 0.66	0.0019
Mystic	-0.1123	x_1: 0.00, x_2: 0.03	0.0219

According to the literature report, this objective function is concave and constraint is convex. Solution of Scipy.optimize does not satisfy the constraint even if it gives the minimum objective value. Therefore, it is not a valid solution for this nonconvex example. Laptimize have obtained the approximated global solution with minimum objective value while maintaining constraint values.

5.2.14 Example 15

Source: [20]

$$Minimize: \quad f_0(x_1, x_2) = f_0(x_1) + f_0(x_2)$$

$$f_0(x_1) = -2x_1 - x_1^2$$

$$f_0(x_2) = x_2$$

$$Subject\ to: \quad f_1(x_1, x_2) = 2x_1 + 3x_2 \leq 6$$

$$f_1(x_1, x_2) = 2x_1 + x_2 \leq 4$$

$$0 \leq x_1 \leq 3, 0 \leq x_2 \leq 4$$

Package	Objective Value	Final Solution	Time(s)
Laptimize	-2.4375	x_1: 0.75, x_2: 1.50	0.0738
scipy.optimize	-5.0000	x_1: 0.99, x_2: 4.00	00.0029
Mystic	-1.1778	x_1: 0.071, x_2: 1.04	00.0428

According to the literature report, this objective function is convex and constraint also a convex function. Solution of Scipy.optimize does not satisfy the constraint even if it gives the minimum objective value. Therefore, it is not a valid solution for this example. Laptimize have obtained the approximated global solution with minimum objective value while maintaining the constraint values.

5.3 Summary

Table 5.13 Package evaluation summary results

Example	Package	Objective value	Solution	Local/Global
1	laptimize	-3.0182	x_1: 1.75, x_2: 1.02	Global
	scipy.optimize	-1.9790	x_1:1.65, x_2: 1.19	Local
	Mystic	-2.9237	x_1:1.65, x_2: 0.91	Local
2	Laptimize	10.00	x_1: 0.00, x_2: 2.00	Global
	scipy.optimize	16.59	x_1: 0.63, x_2: 1.71	Local
	Mystic	12.00	x_1: 1.00, x_2: 0.00	Local
3	Laptimize	2.00	x_1: 0.00, x_2: 0.00	Global
	scipy.optimize	2.00	x_1: 0.00, x_2: 0.00	Global
	Mystic	2.00	x_1: 0.00, x_2: 0.00	Global
4	Laptimize	-8.2255	x_1: 1.97, x_2: 1.50	Global
	scipy.optimize	-2.6189	x_1: 2.00, x_2: 3.87	Local
	Mystic	-8.0032	x_1: 1.82, x_2: 1.31	Global
5	Laptimize	-1.375	x_1: 0.75, x_2: 0.875	Global
	scipy.optimize	-8.7499	x_1: 0.00, x_2: 2.00	Not a solution
	Mystic	-0.9659	x_1: 0.26, x_2: 0.63	Local
6	Laptimize	-23.00	x_1: 2.00, x_2: 3.00, x_3: 0.00	Global
	scipy.optimize	-27.49	x_1: 3.00, x_2: 4.00, x_3: 0.00	Not a solution
	Mystic	-19.68	x_1: 0.94, x_2: 2.65, x_3: 5.00	Local
7	Laptimize	-3.8125	x_1: 0.00, x_2: -0.75, x_3:1.5, x_4: -2.25	Global
	scipy.optimize	-11.4325	x_1: -1.58, x_2: -2.5, x_3: 1.58, x_4: -2.5	Not a solution
	Mystic	-4.6458	x_1:0.5, x_2: -2.24, x_3: 0.83, x_4: -2.47	Not a solution
8	Laptimize	-0.8089	x_1: 0.925, x_2: 0.127	Global
	scipy.optimize	-0.8089	x_1: 0.922, x_2: 0.128	Global
	Mystic	4.99375	x_1: 0.171, x_2: 0.108	Not a solution
9	Laptimize	1430	x_1: 11.04, x_2: 12.0, x_3:1.16	Global
	scipy.optimize	0	x_1: 0.00, x_2: 0.00, x_3:0.00	Not a solution
	Mystic	0	x_1: 0.00, x_2: 0.00, x_3:0.00	Not a solution
10	Laptimize	-106.0	x_1: 200, x_2: 250	Global
	scipy.optimize	-113.0	x_1: 241, x_2: 250	Not a solution
	Mystic	5.0385	x_1: 0., x_2: 250	Not a solution

11	Laptimize	-6.4792	x_1: 2.41, x_2: 2.25	Global
	scipy.optimize	5. 9755	x_1: 3.87, x_2: 0.07	Not a solution
	Mystic	6.2195	x_1: 3. 88, x_2: 0.04	Local
12	Laptimize	-2.8283	x_1: 0.000, x_2: 1.4141	Global
	scipy.optimize	-3.0000	x_1: 0.000, x_2: 1.5	Not a solution
	Mystic	-1.9647	x_1: 1.75, x_2: 0.5448	Local
13	Laptimize	-4.6013	$x = 5.2$	Global
	scipy.optimize	-4.6013	$x = 5.2$	Not a solution
	Mystic	-0.5201	$x = 2.719$	Not a solution
14	Laptimize	-13.6250	x_1: 2.25, x_2: 0.50	Global
	scipy.optimize	-13.8333	x_1: 2.50, x_2: 0.66	Global
	Mystic	-0.1123	x_1: 0.00, x_2: 0.03	Not a solution
15	Laptimize	-2.4375	x_1: 0.75, x_2: 1.50	Global
	scipy.optimize	-5.0000	x_1: 0.99, x_2: 4.00	Not a solution
	Mystic	-1.1778	x_1: 0.071, x_2: 1.04	Not a solution

According to the summary table, laptimize was able to generate global solution for all the problems with its default parameters.

5.4 Implementation Comparison

This section illustrates the implementation format of each package with respect to the Example 01

laptimize

```
 problem = {
    'objective': {'x1': lambda x: 2 * x ** 3 - 9 * x ** 2 + 9 * x,
'x2':  lambda x: -2 * x ** 3 + 9 * x ** 2 - 9 * x,},
    'constraints_1': {'x1': lambda x: -6 * x ** 2, 'x2': lambda x:
18 * x, 'value': 9},
    'constraints_2': {'x1': lambda x: 6 * x ** 2, 'x2': lambda x: -
18 * x, 'value': 0},
    'capacity': {'x1': [0, 3], 'x2': [0, 3]}}

values = Solver(problem).solve()
print(values)
```

scipy.optimize

```
func = lambda x: 2 * x[0] ** 3 - 9 * x[0] ** 2 + 9 * x[0] - 2 * x[1]
** 3 + 9 * x[1] ** 2 - 9 * x[1]


cons = ({'type': 'ineq', 'fun': lambda x: -6 * x[0] ** 2 + 18 * x[1]
- 9},
        {'type': 'ineq', 'fun': lambda x: 6 * x[0] ** 2 - 18 * x[1]
- 0})

result = minimize(func, (0, 0), method='SLSQP', bounds=((0, 3), (0,
3)), constraints=cons)
print(result)
```

Mystic

```
def objective(x):
    return 2 * x[0] ** 3 - 9 * x[0] ** 2 + 9 * x[0] - 2 * x[1] ** 3
+ 9 * x[1] ** 2 - 9 * x[1]

objective_1 = lambda x: objective(x)
equations = '''
 -6 * x0 ** 2 + 18 * x1 - 9 <= 0
 6 * x0 ** 2 - 18 * x1 - 0 <= 0
'''

bounds = [(0, 3), (0, 3)]
eqn =
ms.generate_constraint(ms.generate_solvers(ms.simplify(equations)))
result = diffev2(objective_1, x0=bounds, constraints=eqn,
bounds=bounds, disp=False, full_output=True)

print(result)
```

Problem implementation and parameter setting structure in laptimize implementation is convenient than other two packagers.

5.3 Discussion

Mathematical programming is the fundamental tool of operation research, which is frequently used to find solutions to optimization problems arising in many fields. Also, optimization has become the main component of the data science field in recent decades. While the linear programming model works well in many cases, some problems need nonlinear components to be correctly modeled.

Nonlinear problems are still not harder when it has convex objective and constraints functions. There are many more methods to handle convex non-linear problems as discussed in chapter 01. The presence of non-convexity makes nonlinear problems harder and fewer research methods have been conducted to find optimal global solutions. The branch and bound algorithm that Flank 1972 [5] proposed is guaranteed a global optimum for a large class of nonlinear programming problems in a finite no of steps. Therefore, having a computer program is very useful with this methodology.

As the results of the above requirement, laptimize python package was developed and released to PyPi repository as a publicly available package. The main problem that faced when developing the package was the lack of theoretical materials as well as the complexity of the algorithm. It has proven that laptimize ended up with the global solution for nonlinear, non-convex, separable optimization problems in a finite number of steps.

Further, it was observed that scipy.optimize and mystic results are highly dependent on the initial value of the variables. But laptimize does not need any variable initialization. Out of the examples that considered in the results section laptimize have able to generate a global solution for all of the problems. This will be a plus point for the researchers who are interested in nonlinear optimization problems.

Also, “patition_len” and “error” are the only input parameters to the laptimize, but mystic and scipy.optimize have lot of parameters change (according to the documentation of mystic and scipy.optimize). Users’ needs to experiment with these parameters to find the optimal answer which is time consuming.

Laptimize only can handle additive separable functions as Flak’s algorithm only have considered the additive separable function in algorithm development.

CHAPTER 06
CONCLUTIONS AND POISIBLE FUTURE WORKS

6.1 Conclusions

The conclusions that can be arrived at after the research are stated below.

- laptimize can be used as a reliable python package to solve nonlinear, nonconvex, and separable optimization problems compared to mystic and scipy.optimize without any variable initialization.

- laptimize problem input structure is simpler compared to mystic and scipy.optimize

- partition_len parameter is the most significant and important parameter in the laptimize as it can be used as a tuning parameter to get a more correct solution

6.2 Future Works

The following areas can be considered for future works to improve the scope, correctness, and speed of the algorithm.

- Improving the package to solve non-constraint optimization problems.

- Appling Multiprocessing components to handle branch and bound algorithms and which will reduce the running time.

- Adding more input parameters to get from user, such as upper bound and lower bound which will reduce the no of iterations.

- Adding solution type (ex: minimization, maximization or saddle point) as an output.

References

[1] M. S. Bazaraa, H. D. Sherali and C. M. Shetty, Nonlinear Programming, Theory and Algorithm, John Wiley & Sons, 1993.

[2] J. E. Falk and R. M. Soland, "An Algorithm for Separable Nonconvex Programming Problems," *Management Science*, vol. 15, no. 9, pp. 550-569, 1969.

[3] E. L. Lawler and D. E. Wood, "Branch-And-Bound Methods: A Survey," *Operations Research*, vol. 14, no. 4, pp. 699-719, 1966.

[4] R. M. Soland, "An Algorithm for Separable Nonconvex Programming Problems II: Nonconvex Constraints," *Management Science*, vol. 17, no. 11, pp. 759-773, 1971.

[5] J. E. Falk, "An Algorithm for Locating Approximate Global Solution of Nonconvex Separable Programming," 1972. [Online]. Available: https://apps.dtic.mil/sti/citations/AD0743498.

[6] E. H. L. Beale and J. A. Tomlin, "Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables," in *Fifth International Conference on Operations Research*, London, 1970.

[7] R. M. Soland, "An algorithm for separable piecewise convex programming problems," 1973. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800200213.

[8] H. J. Grotte, *A Computer Program for Solving Separable Nonconvex Optimization Problems*, Program Analysis Division Institute for Defense Analyses, 1978.

[9] L. Han-Lin and Y. Chian-Son, "A global optimization method for nonconvex separable programming problems," 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221798002434.

[10] C. S. Adjiman, I. P. Androulakis and C. A. Floudas, "Global Optimization of Mixed-Integer Nonlinear Problems," *American Institute of Chemical Engineers Journal*, vol. 46, no. 9, 2000.

[11] L. Bao-Liang and I. Koji, "Converting general nonlinear programming problems into separable programming problems with feedforward neural networks," *Neural Networks*, vol. 16, no. 7, pp. 1059-1074, 2003.

[12] X. Hong-gang, X. Cheng-xian and X. Feng-min, "Branch and bound algorithm for separable concave programming," *Journal of Computational Mathematics*, vol. 22, no. 6, p. 895–904, 2004.

[13] T. Kuno and J. Shi, "Linear Programs With an Additional Separable Concave Constraint," *Journal of Applied Mathematics and Decision Sciences*, vol. 8, no. 3, pp. 155-174, 2004.

[14] K. L. Croxton, B. Gendron and T. L. Magnanti, "A Comparison of Mixed-Integer Programming Models for Non-Convex Piecewise Linear Cost Minimization Problems," 2003. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/mnsc.49.9.1268.16570.

[15] M. Stuber, "A Differentiable Model for Optimizing Hybridization of Industrial Process Heat Systems with Concentrating Solar Thermal Power," *Processes,* vol. 6, no. 7, 2018.

[16] [Online]. Available: http://www.universalteacherpublications.com/univ/ebooks/or/Ch15/seprablex1.htm.

[17] M. V. Barkova, " On Generating Non-Convex Optimization Test Problems, Mathematical Optimization Theory and Operations Research," in *Mathematical Optimization Theory and Operations Research,18th International Conference,* 2019.

[18] P. A. Jensen and J. F. Bard, "Nonlinear Programming Methods-Separable Programming," in *Operations Research Models and Methods*.

[19] H. A. L. Thi and M. Ouanes, "Convex quadratic underestimation and Branch and Bound for univariate global optimization with one nonconvex constraint," *RAIRO - Operations Research,* vol. 40, no. 3, pp. 285-302, 2006.

[20] B. K. Patel, "Solutions Of Some Non-Linear Programming Problems," National Institute of Technology, Rourkela, 2014.

[21] N. Agin, *Optimum Seeking with Branch and Bound,* 1966.

[22] M. Tawarmalani and N. V. Sahinidis, "Global optimization of mixed-integer nonlinear programs: A theoretical and computational study," *Mathematical Programming,* vol. 99, no. 3, pp. 563-591, 2004.

[23] A. Yenipazarli, H. P. Benson and S. Erenguc, "A branch-and-bound algorithm for the concave cost supply problem," *International Journal of Production Research,* vol. 54, no. 13, pp. 3943-3961, 2016.

[24] M. D. Stuber, J. K. Scott and P. I. Barton, "Convex and concave relaxations of implicit functions," *Optimization Methods and Software,* vol. 30, no. 3, pp. 424-460, 2014.

[25] S. M. Stefanov, "Convex separable minimization problems with a linear constraint and bounded variables," *International Journal of Mathematics and Mathematical Sciences,* vol. 2005, no. 9, pp. 1339-1363, 2005.

[26] M. J. Box, "A new method of constrained optimization and a comparison with other methods," *The Computer Journal,* vol. 8, p. 42, 1965.

[27] [Online]. Available: https://apps.dtic.mil/sti/citations/AD0743498.

[28] B. Patel, "Solutions Of Some Non-Linear Programming Problems," 2014.

Appendix A

SERIAL T-262

AN ALGORITHM FOR LOCATING APPROXIMATE GLOBAL SOLUTIONS OF NONCONVEX, SEPARABLE PROBLEMS

James E. Falk

April 20, 1972

THE GEORGE WASHINGTON UNIVERSITY
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
INSTITUTE FOR MANAGEMENT SCIENCE AND ENGINEERING
Program in Logistics

1. INTRODUCTION

An algorithm for finding global solutions of nonconvex separable problems was developed by Falk and Soland [3] and Soland [8]. The method is based on the branch and bound philosophy and yields a (generally infinite) sequence of points whose cluster points are global solutions of the problem. The implementation of the method is severely limited by the necessity of computing convex envelopes [4] of the functions involved although a number of applications of the method have been made (e.g., [5], [9]). These applications were possible because of the special structure of the functions involved (e.g., concave or piecewise linear).

The traditional method for treating separable problems involves calculating piecewise linear approximations of the functions defining the problem and applying a modification of the simplex method to the resulting problem (see, e.g., Miller [7]). The modification amounts to a restriction on the usual manner of selecting variables to exchange roles (basic to nonbasic and vice versa) and will yield a local but not necessarily a global solution of the approximating problem.

In this paper we present a method that will yield a global solution of the approximating problem referred to above. The method is similar to the Falk-Soland algorithm but takes advantage of the special structure of the resulting approximate problem and employs the branch and bound philosophy to set up and monitor the solutions of a finite sequence of linear subproblems.

Recently Beale and Tomlin [1] announced that they have developed a similar algorithm which they have incorporated into their UMPIRE mathematical programming system [10]. The basic idea of their method is the same as that of the algorithm detailed herein although their rules for selecting branching nodes and branching variables are different, being developed from an integer programming point of view while ours are modifications of the rules developed in the Falk-Soland method [3] and its extension by Soland [8].

The problem which we address has the form

$$\text{problem Q} \begin{cases} \text{minimize} & F_o(x) \\ \text{subject to} & F_i(x) \leq b_i \quad i = 1,\ldots,m \\ & \ell \leq x \leq L \end{cases}$$

where ℓ and L are finite lower and upper bounds respectively on x. We assume that each F_i $(i=0,1,2,\ldots,m)$ is separable, i.e.,

$$F_i(x) = \sum_{j=1}^{n} F_{ij}(x_j) \quad i = 0,1,\ldots,m$$

and that each F_{ij} is continuous. As extension to the case where F_{ij} is piecewise continuous is covered in Section 5.

In Section 2 we define the approximating problem of problem Q and construct the problem obtained by replacing each of the functions involved by their convex envelopes. A related problem is simultaneously introduced and shown to give a sharper underestimate of the optimal value of the approximating problem than does the convex envelope problem. It is this related problem which the branch and bound procedure solves first to get estimates on the optimal value of the approximating problem and to set up new problems if the estimates do not yield a global solution.

A detailed analysis of the complete method is given in Section 3 and an example follows in Section 4. Some computational considerations are given in Section 5.

2. THE APPROXIMATING PROBLEM AND CONVEX ENVELOPES

The approximating problem of the original problem Q is obtained by replacing each function F_{ij} by a piecewise linear approximation over the interval $[\ell_j, L_j]$. One common method (see, e.g., [7]) that is employed involves selecting $p_j + 1$ grid points $y_{j0}, \ldots, y_{jp_j}$ in $[\ell_j, L_j]$ where $y_{j0} = \ell_j$ and $y_{jp_j} = L_j$ and using convex combinations of the numbers $F_{ij}(y_{jk})$ and $F_{ij}(y_{j,k+1})$ as approximations to the values of $F_{ij}(x_j)$ over the subinterval $[y_{jk}, y_{j,k+1}]$. Figure 1 illustrates this type of approximation.

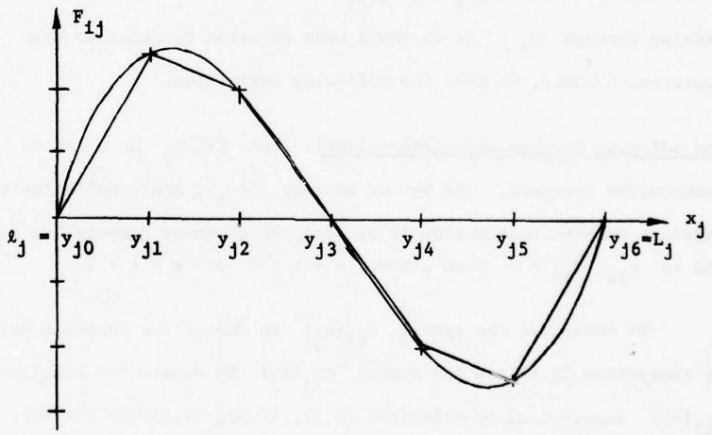


Figure 1. PIECEWISE LINEAR APPROXIMATIONS

Mathematically, we obtain this approximation by setting $F_{ij}(x_j) \tilde{=} f_{ij}(\theta_j)$ where

$$f_{ij}(\theta_j) = \sum_{k \in K_j} \theta_{jk} F_{ij}(y_{jk}) \tag{2.1}$$

where $K_j = \{0,1,\ldots,p_j\}$, $\theta_j = (\theta_{j0},\ldots,\theta_{jp_j})$ if

$$\sum_{k \in K_j} \theta_{jk} y_{jk} = x_j \tag{2.2}$$

$$\sum_{k \in K_j} \theta_{jk} = 1 \tag{2.3}$$

$$\theta_{jk} \geq 0 \qquad k \in K_j \tag{2.4}$$

and if we add the further restriction that at most two of the weights $\{\theta_{jk} : k \in K_j\}$ are nonzero, and if two are nonzero, then these must correspond to adjacent grid points. This last restriction is necessary since without it one may obtain any point in the convex hull of the set $\{(y_{j0},F_{ij}(y_{j0})),\ldots(y_{j,p_j},F_{ij}(y_{j,p_j}))\}$ which lies on the vertical line passing through x_j. As we shall have occasion to refer to this restriction later, we make the following definition.

The Adjacent Weights Restriction (AWR): Let $K \subset K_j$ be a set of consecutive integers. The set of numbers $\{\theta_{jk} : k \in K\}$ satisfies the adjacent weights restriction if at most two of these numbers are nonzero, and if $\theta_{js}, \theta_{jt} > 0$ then either $s = t - 1$ or $s = t + 1$.

We shall use the symbol $f_{ij}(\theta_j)$ to denote the function defined by expression (2.1) and the symbol $f_{ij}(x_j)$ to denote the function $f_{ij}(\theta_j)$ constrained by relations (2.2), (2.3), (2.4) and the AWR. Thus $f_{ij}(\theta_j)$ denotes a linear function of the variables $\theta_{j0},\theta_{j1},\ldots,\theta_{jp_j}$ while $f_{ij}(x_j)$ denotes a piecewise linear function

of the single variable x_j such as that illustrated in Figure 1.

Likewise

$$f_i(\theta) = \sum_{j=1}^{n} f_{ij}(\theta_j)$$

and

$$f_i(x) = \sum_{j=1}^{n} f_{ij}(x_j)$$

for $i = 0,1,2,\ldots,m$.

By replacing each $F_{ij}(x_j)$ by its piecewise linear approximation $f_{ij}(x_j)$, we obtain the following approximate problem

$$\text{problem P} \begin{cases} \text{minimize} & f_o(\theta) = \sum_{j=1}^{n} \sum_{k\in K_j} \theta_{jk} F_{oj}(y_{jk}) \\ \text{subject to} & f_i(\theta) = \sum_{j=1}^{n} \sum_{k\in K_j} \theta_{jk} F_{ij}(y_{jk}) \leq b_i & (i = 1,\ldots,m) \\ & \sum_{k\in K_j} \theta_{jk} = 1 & (j = 1,\ldots,n) \\ & \theta_{jk} \geq 0 & (j = 1,\ldots,n\ ;\ k\in K_j) \\ & \{\theta_{jk} : k\in K_j\} \text{ satisfies AWR} & (j = 1,\ldots,n) \end{cases}$$

Here $\theta = (\theta_1;\theta_2;\ldots;\theta_n) = (\theta_{10},\ldots,\theta_{1p_1};\theta_{20},\ldots;\ldots;\theta_{n0},\ldots,\theta_{np_n})$.
The solution <u>value</u> of this problem is offered as an approximation to the solution value of the original problem, problem Q. The solution <u>point</u> θ^* of problem P yields an approximation to the solution of problem Q via the relations (2.2), i.e.,

$$x^*_j = \sum_{k\in K_j} \theta^*_{jk} y_{jk} \qquad j = 1,\ldots,n .$$

Problem P is the usual problem that is addressed when seeking solutions of separable programs (see, e.g., [7]). The method of "solution" involves generating a basic feasible solution of the linear

program associated with problem P that satisfies the AWR. A modification of the simplex method is then used to sequentially change the basis until a local solution of problem P is obtained. This modification amounts to a restricted basis entry rule which insures that the AWR are always satisfied by the basic feasible solution associated with each stage of the simplex method. Thus the only nonbasic variables θ_{jk} that may enter the basis at a given iteration are neighbors of existing basic variables. If such a variable is chosen to enter the basis, the outgoing basic variable must be chosen so that the new basic feasible solution satisfies the AWR. It may be shown that this method will yield a local solution of problem P, so that if problem P is convex, the solution will be a global solution. In particular, if problem Q is convex, then so is P and a global solution is assured.

In this paper we are concerned with a method that will produce <u>global</u> solutions of problem P. The method may be considered a specialization of the method of Falk and Soland [3] and the extension described by Soland [8]. In this method it is necessary to compute "convex envelopes" of all functions involved in the problem description over appropriate intervals. A number of convex subproblems are then set up and solved with the branch and bound philosophy monitoring the solution values of these problems and guiding the creation of new subproblems. The <u>convex envelope</u> of a function of a single variable $f_{ij}(x_j)$ over an interval $[\ell_j, L_j]$ is that convex function f_{ij}^c defined over $[\ell_j, L_j]$ such that, if d_{ij} is any convex function on $[\ell_j, L_j]$ which underestimates f_{ij} at every point in $[\ell_j, L_j]$, then d_{ij} also underestimates f_{ij}^c over $[\ell_j, L_j]$. Roughly, the convex envelope of a function is the highest convex function which underestimates that function over the appropriate interval. Alternate and more general definitions and relations concerning convex envelopes are found in [4].

We are interested in determining the convex envelope of the piecewise linear functions $f_{ij}(x_j)$ defined by the relations (2.1) through (2.4) together with the AWR. It is clear geometrically, and not difficult to show analytically, that the convex envelope of this function over $[\ell_j, L_j]$ is the function $f_{ij}^c(x_j)$:

$$f_{ij}^c(x_j) = \min_{\theta_j} \sum_{k\epsilon K_j} \theta_{jk} F_{ij}(y_{jk}) \tag{2.5}$$

$$\text{s.t.} \quad \sum_{k\epsilon K_j} \theta_{jk} y_{jk} = x_j \tag{2.6}$$

$$\sum_{k\epsilon K_j} \theta_{jk} = 1 \tag{2.7}$$

$$\theta_{jk} \geq 0, \; k\epsilon K_j \; . \tag{2.8}$$

Note that we do not impose the AWR on the definition of $f_{ij}^c(x_j)$. We illustrate this definition in Figure 2 which may be compared to Figure 1.

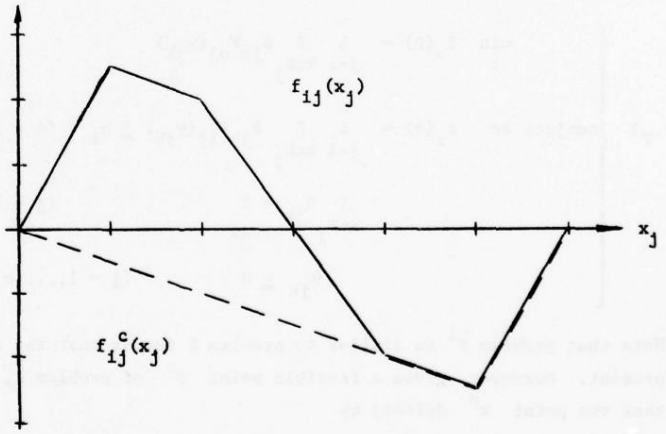


Figure 2. CONVEX ENVELOPES

A-7

Thus the calculation of $f_{ij}^c(x_j)$ at a given point x_j involves the solution of a linear program. The first subproblem addressed by the method described in [8] would be

$$\min \quad f_o^c(x) = \sum_{j=1}^{n} f_{oj}^c(x_j)$$

$$\text{subject to} \quad f_i^c(x) = \sum_{j=1}^{n} f_{ij}^c(x_j) \leq b_i \qquad (i = 1,\ldots,m)$$

$$\ell \leq x \leq L \ .$$

This is a convex program whose solution value serves as an underestimate of the solution value of problem P. Because of the piecewise linear nature of the functions f_{ij}^c, it is possible to convert this problem to a linear program. This approach, however, involves explicitly calculating the functions f_{ij}^c for each i and j. Moreover, it would be necessary to do this for a number of problems of the above form. We may avoid these calculations by considering the related linear program:

$$P^1 \begin{cases} \min\limits_{\theta} \quad f_o(\theta) = \sum\limits_{j=1}^{n} \sum\limits_{k\in K_j} \theta_{jk} F_{oj}(y_{jk}) \\ \text{subject to} \quad f_i(\theta) = \sum\limits_{j=1}^{n} \sum\limits_{k\in K_j} \theta_{jk} F_{ij}(y_{jk}) \leq b_i \qquad (i = 1,\ldots,m) \\ \qquad\qquad \sum\limits_{k\in K_j} \theta_{jk} = 1 \qquad (j = 1,\ldots,n) \\ \qquad\qquad \theta_{jk} \geq 0 \qquad (j = 1,\ldots,n \ ; \ k\in K_j) \ . \end{cases}$$

Note that problem P^1 is similar to problem P except that the AWR are not present. Moreover, given a feasible point θ^o of problem P, it follows that the point x^o defined by

$$x_j^o = \sum_{k\in K_j} \theta_{jk}^o y_{jk}$$

is feasible for the convex envelope problem by virtue of the inequality

$$f_{ij}^{c}(x_j^o) \leq f_{ij}(\theta^o) .$$

It is, however, possible that the convex envelope problem has feasible points x for which there is no feasible θ satisfying the above expression. For if x is feasible to the convex envelope problem, for each $i = 1,\ldots,m$ there must be a vector i which satisfies conditions (2.6), (2.7), and (2.8) together with the conditions $f_i(\theta) \leq b_i$. This, in itself, does not imply the existence of a single vector which satisfies all of these conditions. On the other hand, any point feasible to problem P is also feasible to P^1 so that the solution value of P^1 offers a valid lower bound on the solution value of P.

3. THE BRANCH AND BOUND ALGORITHM

In this section we present an algorithm to calculate the global solution of problem P which is based on the branch and bound philosophy (see, e.g., [6]). The algorithm considers subsets of a linear polyhedron containing the feasible region $F(P)$ of problem P. A lower bound on the optimal value of problem P is found by minimizing $f_o(\theta)$ over each of these subsets and selecting the smallest of these. A check for solution is made which, if successful, yields a global solution of P. If the check fails, the subset corresponding to the smallest lower bound is further subdivided into either two or three new linear polyhedra and the process continues as before with new and sharper bounds being determined. The process is finite and terminates with a global solution of P.

As is customary with branch and bound procedures, the algorithm is described in terms of a branch and bound tree. (See Figure 6 for an example.) The nodes of the tree will be identified with the symbols N^1, N^2, N^3,... and each node N^i will correspond to a linear subproblem P^i of problem P. It is convenient to also use the notion of a "stage." The first stage of the method consists of problem P^1 (or node N^1) and

its solution. The second stage of the algorithm consists of problems P^1 together with either 2 or 3 new subproblems created from problem P^1. A new stage is created when a previously solved subproblem is chosen for branching and new subproblems are formed. For example, the tree of Figure 6 illustrates that 8 subproblems were formed in 4 stages. The first stage contains node N^1; the second contains nodes N^1, N^2, N^3 and N^4; the third stage contains these nodes and the new nodes N^5 and N^6, and the fourth stage contains nodes N^1 through N^8.

With each node N^t there is associated a linear program of the form

$$\text{problem } P^t \begin{cases} \text{minimize} & f_o(\theta) = \sum_{j=1}^{n} \sum_{k\in K_j} \theta_{jk} F_{oj}(y_{jk}) & \\ \text{subject to} & f_i(\theta) = \sum_{j=1}^{n} \sum_{k\in K_j} \theta_{jk} F_{ij}(y_{jk}) \leq b_i & (i=1,\ldots,m) \\ & \sum_{k\in K_j} \theta_{jk} = 1 & (j=1,\ldots,n) \\ & \theta_{jk} \geq 0 & (j=1,\ldots,n;\ k\in K_j) \\ & \theta_{jk} = 0 & (j=1,\ldots,n;\ k\notin K_j^t) \end{cases}$$

where the sets K_j^t $(j=1,\ldots,n)$ are subsets of consecutive integers of the sets K_j. Note that each problem is a linear program and that these problems differ only in the constraints $\theta_{jk} = 0$ $(j=1,\ldots,n;\ k\notin K_j^t)$. Problem P^1 has $K_j^1 = K_j$ $(j=1,\ldots,n)$ so that problem P^1 resembles problem P except that problem P^1 does not have the AWR imposed on it. Let $F(P^t)$ denote the feasible region of problem P^t and $F(P)$ denote the feasible region of problem P. Note that

$$F(P) \subset F(P^1) \qquad (3.1)$$

and that $F(P^1)$ is a linear polydedron whereas, in general, $F(P)$ is not even a convex set. Assuming $F(P) \neq \emptyset$, problem P^1 will have at least one minimizing point θ^1. In general, let θ^t denote a solution of problem P^t, if one exists, and set

$$LB(t) = \begin{cases} f_o(\theta^t) & \text{if } \theta^t \text{ exists} \\ +\infty & \text{otherwise.} \end{cases}$$

It follows that

$$LB(t) \leq \min \{f_o(\theta) \ : \ \theta \epsilon F(P^t) \cap F(P)\} . \qquad (3.2)$$

It is sometimes possible to obtain an upper bound on $f_o(\theta^*)$ from problem P^t. In fact, if $\hat{\theta}$ is any feasible point to problem P, the number $f_o(\hat{\theta})$ will be an upper bound on $f_o(\theta^*)$. Using the vector θ^t (assuming it exists) we may, at little computational expense, attempt to construct a vector $\hat{\theta}^t$ which is feasible to problem P according to the following rule:

Compute the vector x^t using the relationship

$$x_j^t = \sum_{k \epsilon K_j} \theta_{jk}^t y_{jk} \qquad (j=1,\ldots,n) .$$

We then compute a vector $\hat{\theta}^t$ which satisfies the AWR and the relationship

$$x_j^t = \sum_{k \epsilon K_j} \hat{\theta}_{jk}^t y_{jk} \qquad (j=1,\ldots,n) .$$

This computation is straightforward since each x_j^t must be in some interval $[y_{j,k'}, y_{j,k'+1}]$ and hence may be expressed as a convex combination of the two adjacent points $y_{j,k'}$ and $y_{j,k'+1}$. If this vector $\hat{\theta}^t$ also satisfies the constraints $f_i(\theta) \leq b_i$ $(i=1,\ldots,m)$, the number $f_o(\hat{\theta}^t)$ serves as an upper bound on $f_o(\theta^*)$. We define

the quantity

$$UB(t) = \begin{cases} f_o(\hat{\theta}^t) & \text{if } \hat{\theta}^t \text{ is feasible to P} \\ +\infty & \text{otherwise} \end{cases}$$

so that

$$f_o(\theta^*) \leq UB(t) \tag{3.3}$$

serves as a complementary inequality to (3.2).

In general, the ℓ-th stage of the algorithm consists of problems $P^1,\ldots,P^L$ together with their solutions $\theta^1,\ldots,\theta^L$ (if they exist) and the quantities $LB(1), UB(1),\ldots,LB(L), UB(L)$. A node (or equivalently, a problem) from which no branching has yet taken place (from which no new problems have been created) is termed an <u>intermediate node</u> (<u>intermediate problem</u>). The set of all intermediate problems at stage ℓ is denoted by $I(\ell)$. At stage one, $I(\ell) = \{1\}$, and, if three new problems are created to form stage two, $I(2) = \{2,3,4\}$.

The algorithm is to be constructed in such a way that

$$F(P) \subset \bigcup_{t\in I(\ell)} F(P^t) \ . \tag{3.4}$$

We define the quantities

$$BLB(\ell) = \min_{t\in I(\ell)} \{LB(t)\}$$

and

$$BUB(\ell) = \min_{t=1,\ldots,L} \{UB(t)\} \ .$$

Then (3.2), (3.3) and (3.4) imply that

$$BLB(\ell) \leq f_o(\theta^*) \leq BUB(\ell) \ . \tag{3.5}$$

This is the basic inequality which signals the completion of the algorithm when equality is attained throughout. We will show that our method of branching (creating new problems) sequentially sharpens (3.5) stage by stage and will produce equality in a finite number of stages.

Check for Solution: If $BLB(\ell) = BUB(\ell)$ at the ℓ-th stage, an optimal solution of problem P is $\overset{\gamma t}{\theta}$ where $UB(t) = f_o(\overset{\gamma t}{\theta}) = BUB(\ell)$.

If $BLB(\ell) < BUB(\ell)$ we must choose a node N^t for branching, i.e., a problem P^t to create new problems which will sharpen the bounds in (3.5). We shall use the notion that the numbers $LB(t)$ represent approximations to the quantities $\min \{f_o(\theta) : \theta \epsilon F(P^t) \cap F(P)\}$. Since we are interested in determining $\min \{f_o(\theta) : \theta \epsilon F(P)\}$, we choose the smallest of the numbers $LB(t)$ to determine P^t, the problem most likely to generate a global solution of P.

Choice of Branching Node: Choose an intermediate node N^T for further branching where $LB(T) = BLB(\ell)$.

Actually the algorithm will converge if any intermediate node is selected for further branching and it is sometimes convenient from a computational point of view to use a different rule for branching. A common alternative is to select that problem which has been solved last for further analysis, since the data defining that problem are on hand and data needed for the new problem are very similar. This alleviates the bookkeeping involved and tends to minimize the number of times a particular branch in the tree is revisited. On the other hand, the tree tends to grow larger than the tree our rule would grow and would not be efficient if the total time required is largely a function of the time required to solve the subproblems. In our application, the amount of data required to distinguish one problem from another is minimal so that this should not be a factor.

Having selected node N^T for branching at stage ℓ, we create new subproblems by choosing a branching variable θ_J (or, equivalently, x_J) and partitioning the set K_J^T into subsets of consecutive integers. The rule for selecting x_J follows.

Choice of a Branching Variable: Compute each of the differences

$$\sum_{k\in K_j^T} (\hat{\theta}_{jk}^T - \theta_{jk}^T) F_{ij}(y_{jk}) \qquad (3.6)$$

for $i = 0,1,\ldots,m$ and $j = 1,\ldots,n$. Select J which corresponds to the largest of these differences.

If all of these differences were nonpositive, upon summing over j for each $i = 0,1,\ldots,m$ we obtain

$$f_i(\hat{\theta}^T) - f_i(\theta^T) \leq 0 \qquad (i=0,\ldots,m) .$$

Thus

$$f_o(\hat{\theta}^T) \leq f_o(\theta^T) = BLB(\ell)$$

and

$$f_i(\hat{\theta}^T) \leq f_i(\theta^T) \leq b_i \qquad (i=1,\ldots,m)$$

Since $\hat{\theta}^T$ satisfies the AWR, we see that $\hat{\theta}^T \in F(P)$ so that

$$BUB(\ell) \leq f_o(\hat{\theta}^T) \leq BLB(\ell)$$

that is, $\hat{\theta}^T$ must have been a global solution of problem P, contradicting our previous assumption. Thus, unless we are at a solution, at least one of the differences (3.6) is positive and we choose J corresponding to the largest of these quantities.

This rule for selecting a branching variable is analogous to the rule suggested in [3] and [8]. Since, at a solution, all differences (3.6) will be nonpositive, we are selecting a variable corresponding to the worst violation of this criterion.

Note that not all differences (3.6) need be calculated at every stage since some will automatically be zero. If the set $\{\theta_{jk} : j \in K_j^T\}$ satisfies the AWR, for some j then $\theta_j^T = \hat{\theta}_j^T$ so that all of the corresponding differences (3.6) for $i = 0,\ldots,m$ are zero. Moreover,

if $F_{ij}(x_j)$ is a convex function, the piecewise linear approximation $f_{ij}(x_j)$ of it (equations (2.1) through (2.4) and the AWR) will also be convex. If we denote this approximation by $\overset{\gamma}{F}_{ij}(x_j)$ we have

$$\sum_{k\varepsilon K_j} \overset{\gamma}{\theta}{}^{T}_{jk} F_{ij}(y_{jk}) = \sum_{k\varepsilon K_j} \overset{\gamma}{\theta}{}^{T}_{jk} \overset{\gamma}{F}_{ij}(y_{jk})$$

$$= \overset{\gamma}{F}_{ij}\Big(\sum_{k\varepsilon K_j} \overset{\gamma}{\theta}{}^{T}_{jk} y_{jk}\Big) \qquad (\overset{\gamma}{\theta}{}^{T}_{j} \text{ satisfies AWR})$$

$$= \overset{\gamma}{F}_{ij}\Big(\sum_{k\varepsilon K_j} \theta^{T}_{jk} y_{jk}\Big)$$

$$\leq \sum_{k\varepsilon K_j} \theta^{T}_{jk} F_{ij}(y_{jk})$$

so that the corresponding differences (3.6) are automatically nonpositive. Incidentally, this also proves that the algorithm yields a global solution of a convex program in a single stage.

Having selected variable J for branching, we now are in a position to create the new problems of the $(\ell+1)$-st stage. Let $K^T_J = \{p,p+1,\ldots,q,\ldots,r\}$. Note $x^T_J \neq y_{Jp}$ since in this case $\theta^T_{Jp} = 1$ while $\theta^T_{J,p+1} = \ldots = \theta^T_{Jr} = 0$ and the difference (3.6) would be zero. Likewise $x^T_J \neq y_{Jr}$. Note also that K^T_J contains at least three indices for otherwise branching could not take place on this variable. We may assume that $x^T_J \varepsilon [y_{Ja}, y_{Jb}]$ where y_{Ja} is the nearest left neighboring division point of x^T_J and y_{Jb} is the nearest right division point. We do not exclude the case where $x^T_J = y_{Ja} = y_{Jb}$, i.e., where x^T_J falls on a division point.

Recall that problems $P^1,\ldots,P^L$ have been set up and solved at the end of stage ℓ.

Branching Rule (refer to Figure 3):

$$\text{Let } K_J^- = \{k : k\epsilon K_J^T \text{ and } y_{Jk} \leq y_{Ja}\}$$

$$K_J^o = \{a,b\}$$

$$K_J^+ = \{k : k\epsilon K_J^T \text{ and } y_{Jb} \leq y_{Jk}\}$$

Referring to the general definitions of problem P^t at the beginning of this section, define a new problem P^t by setting K_J^t equal to one of the above sets if that set contains at least two elements. The other index sets K_j^t are unchanged (i.e., $K_j^T = K_j^T$ $(j\neq J)$). In this manner we may define at least two new problems (since K_J^T had at least three points and $y_{Jp} \neq x_J^T$, $y_{J\sim} \neq x_J^T$) and possibly three new problems. These problems are numbered P^{L+1}, P^{L+2} and P^{L+3} (if defined). Note that if $a \neq b$, the problem whose index set $K_J^t = \{a,b\}$ must have only solutions with θ_J^t satisfying the AWR. The various possibilities are illustrated by example in Figure 3. Only the first possibility yields three new problems.

A-16

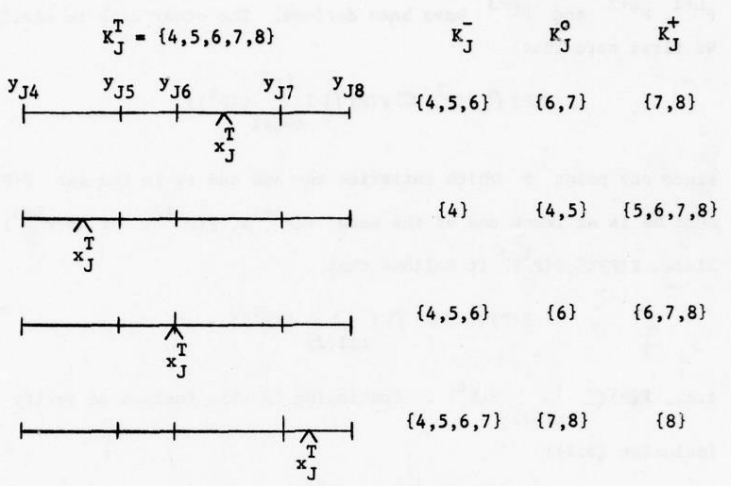


Figure 3. BRANCHING ON VARIABLE x_J

Beale and Tomlin suggest a different branching rule wherein two new subproblems are defined at each stage. Using the above notation, they set

$$K_J^- = \{k : k \epsilon K_J^T \text{ and } y_{Jk} \leq y_{Jb}\}$$

$$K_J^+ = \{k : k \epsilon K_J^T \text{ and } y_{Ja} \leq y_{Jk}\}$$

so that the feasible regions of their problems P^{k+1} and P^{k+2} overlap somewhat more than ours do. Referring to Figure 3, their sets K_J^- and K_J^+ would be $\{4,5,6,7\}$ and $\{6,7,8\}$ in the first case while in the other three cases, their sets would define the same subproblems as we do.

In the remarks which follow we shall assume that these problems P^{L+1}, P^{L+2} and P^{L+3} have been defined. The other case is similar. We first note that

$$F(P) \cap F(P^T) \subset F(P) \cap \left(\bigcup_{t=L+1}^{L+3} F(P^t) \right)$$

since any point θ which satisfies the AWR and is in the set $F(P^T)$ must be in at least one of the sets $F(P^{L+1})$, $F(P^{L+2})$ or $F(P^{L+3})$. Since $F(P) \subset F(P^1)$ it follows that

$$F(P) \subset F(P) \cap \left(\bigcup_{t \in I(2)} F(P^t) \right)$$

i.e., $F(P) \subset \bigcup_{t \in I(2)} F(P^t)$. Continuing in this fashion we verify inclusion (3.4):

$$F(P) \subset \bigcup_{t \in I(\ell)} F(P^t) \tag{3.4}$$

Moreover, since any point in one of the sets $F(P^{L+1})$, $F(P^{L+2})$ or $F(P^{L+3})$ must lie in $F(P^T)$ we have

$$\bigcup_{t \in I(\ell)} F(P^t) \subset \bigcup_{t \in I(\ell-1)} F(P^t) .$$

This inclusion must be strict since the point θ^T cannot lie in any of the sets $F(P^{L+1})$, $F(P^{L+2})$ or $F(P^{L+3})$. For suppose $\theta^T \in F(P^{L+1})$ and $K_J^{L+1} = \{p, \ldots, a\}$. Then $\theta_{Jk}^T = 0$ for $k = a+1, \ldots, r$ and $x_J^T < y_{Ja}$ which contradicts the assumption that $x_J^T \in [y_{Ja}, y_{Jb}]$.

These remarks yield

$$F(P) \subset \bigcup_{t \in I(\ell)} F(P^t) \subsetneq \bigcup_{t \in I(\ell-1)} F(P^t) \subsetneq \ldots \subsetneq F(P^1) \tag{3.7}$$

i.e., the sets $\bigcup_{t\in I(\ell)} F(P^t)$ are converging monotonically towards the set $F(P)$.

When new problems are created for the $(\ell+1)$-st stage, new lower and upper bounds are calculated. Note that

$$\min \{LB(P^{L+1}), LB(P^{L+2}), LB(P^{L+3})\} \geq LB(P^T)$$

since $F(P^T) \not\supseteq \bigcup_{t=L+1}^{L+3} F(P^t)$. Moreover, since the point θ^T for which $f_o(\theta^T) = LB(P^T)$ is not feasible for the new problems, it is likely that the above inequality is strict. The above inequality, together with the definitions of $BLB(\ell)$ and $BUB(\ell)$ yield

$$BLB(1) \leq \ldots \leq BLB(\ell) \leq f_o(\theta^*) \leq BUB(\ell) \leq \ldots \leq BUB(1) \qquad (3.8)$$

so that the upper and lower bounds are converging towards the optimal value of P. It remains to show that the process converges in a finite number of stages.

Theorem. After a finite number of stages, the algorithm yields a global solution of problem P.

Proof. At each stage of the algorithm an index $J\in\{1,\ldots,N\}$ is selected and the set K_J^T is subdivided into either two or three new sets of consecutive integers according to the branching rule. Each of these new sets contains at least two integers. Since there are but a finite number of choices for J and a finite number of ways of subdividing the original sets K_j into sets containing at least two consecutive integers, the algorithm would (if it continued) eventually produce problems whose feasible regions contained only points which satisfy AWR (i.e., eventually $F(P) = \bigcup_{t\in I(\ell)} F(P^t)$). Such problems must be intermediate problems since their regions cannot be further decomposed, and $LB(t) = UB(t)$. Thus equality must eventually occur in (3.8) and the algorithm is finite.

4. AN EXAMPLE

Problem Q:

$$\text{minimize} \quad F_o(x) = (2x_1^3 - 9x_1^2 + 9x_1) + (-2x_2^3 + 9x_2^2 - 9x_2)$$

$$\text{subject to} \quad F_1(x) = -6x_1^2 + 18x_2 \leq 9$$

$$F_2(x) = 6x_1^2 - 18x_2 \leq 0$$

$$0 \leq x_1, x_2 \leq 3$$

The feasible region of this problem is sketched in Figure 4. There are local solutions near the points (0,0.5), (1.787,1.065) and (2.738,3.000) with values -2.50, -2.97 and -1.46 respectively. The subdivision points are taken at intervals of 1/2 starting at 0. These values and the values of functions F_{ij} at these points are displayed in Table 1 and the results of linear approximations are sketched in Figure 5.

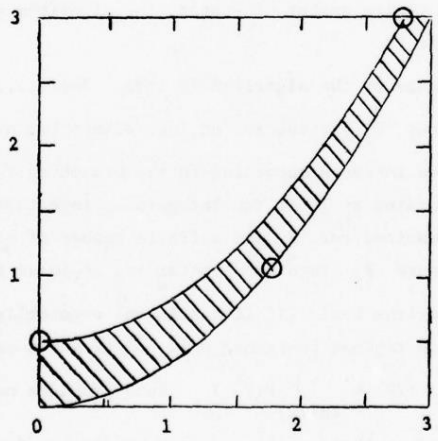
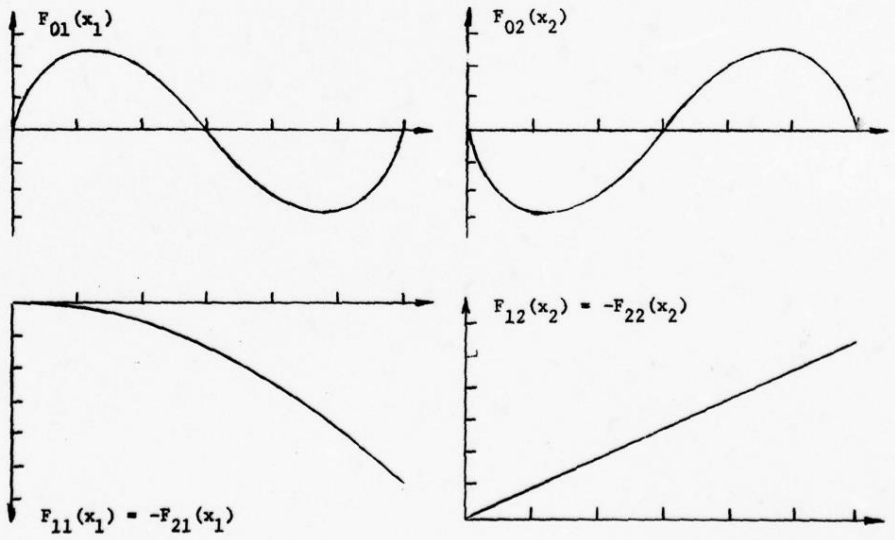


Figure 4. FEASIBLE REGION FOR EXAMPLE

Table 1.
DATA FOR EXAMPLE

x_{1k}	F_{01}	F_{11}	F_{21}	x_{2k}	F_{02}	F_{12}	F_{22}
0	0	0	0	0	0	0	0
1/2	5/2	-3/2	3/2	1/2	-5/2	9	-9
1	2	-6	6	1	-2	18	-18
3/2	0	-27/2	27/2	3/2	0	27	-27
2	-2	-24	24	2	2	36	-36
5/2	-5/2	-75/2	75/2	5/2	5/2	45	-45
3	0	-54	54	3	0	54	-54



A-21

Each subproblem has variable $\theta = (\theta_1,\theta_2) = (\theta_{10},\ldots,\theta_{16};$ $\theta_{20},\ldots,\theta_{26})$. The data provided by subproblems is given in Table 2 and the branch and bound tree is illustrated in Figure 6. The global solution of the approximate problem is found to be the point

$$x^* = (1.714,1.000)$$

with objective function value -2.857. This solution is actually found at node 6 but not recognized until problem 8 has been solved.

Table 2.
SOLUTION VALUES FOR EXAMPLE

stage	problem	index sets	solution	solution value	vector x	vector θ'	objective function	branching variable	best bounds
ℓ	P^t	K_1^t K_2^t	θ_1^t θ_2^t	$f_o(\theta^t)=$ $LB(t)$	x_1^t x_2^t	$\tilde{\theta}_1^t$ $\tilde{\theta}_2^t$	$f_o(\tilde{\theta}^t)=$ $UB(t)$	x_J	$BLB(k)$ $BUB(k)$
1	1	{0,1,2,3,4,5,6}	0 0 0 0 0 1 0 0	-3.667	2.000	0 0 0 0 1 0 0	-2.667	x_2	-3.667
		{0,1,2,3,4,5,6}	0 0 $^5/_6$ 0 0 0 0 $^1/_6$		1.333	0 0 $^1/_3$ $^2/_3$ 0 0 0			-2.667
2	2	{0,1,2,3,4,5,6}	$^1/_4$ 0 0 0 0 $^3/_4$ 0 0	-3.500	1.500	0 0 0 1 0 0 0	-2.000	x_1	
		{0,1,2}	0 0 1 * * * * *		1.000	0 0 1 0 0 0 0			
	3	{0,1,2,3,4,5,6}	$^1/_4$ 0 0 0 0 $^3/_4$ 0 0	-3.500	1.500	0 0 0 1 0 0 0	-2.000	x_1	
		{2,3}	* * 1 0 * * * *		1.000	0 0 1 0 0 0 0			
	4	{0,1,2,3,4,5,6}	0 0 0 0 0 0 1 0	-2.500	2.500	0 0 0 0 0 0 1 0	-0.416	x_2	-3.500
		{3,4,5,6}	* * * $^{11}/_{18}$ 0 0 $^7/_{18}$		2.083	0 0 0 0 $^5/_6$ $^1/_6$ 0			-2.667
3	5	{0,1,2,3}	1 0 0 0 * * * *	-2.500	0.000	$\tilde{\theta}^5 = \theta^5$	-2.500	—	
		{0,1,2}	0 1 0 * * * * *		0.500				
	6	{3,4,5,6}	* * * $^4/_7$ $^3/_7$ 0 0	-2.857	1.714	$\tilde{\theta}^6 = \theta^6$	-2.857	—	-3.500
		{0,1,2}	0 0 1 * * * * *		1.000				-2.857
	7	{0,1,2,3}	0 0 0 1 * * * *	-2.000	1.500	$\tilde{\theta}^7 = \theta^7$	-2.000	—	
		{2,3}	* * 1 0 * * * *		1.000				
4	8	{3,4,5,6}	* * * $^4/_7$ $^3/_7$ 0 0	-2.857	1.714	$\tilde{\theta}^8 = \theta^8$	-2.857	—	-2.857
		{2,3}	* * 1 0 * * * *		1.000				-2.857

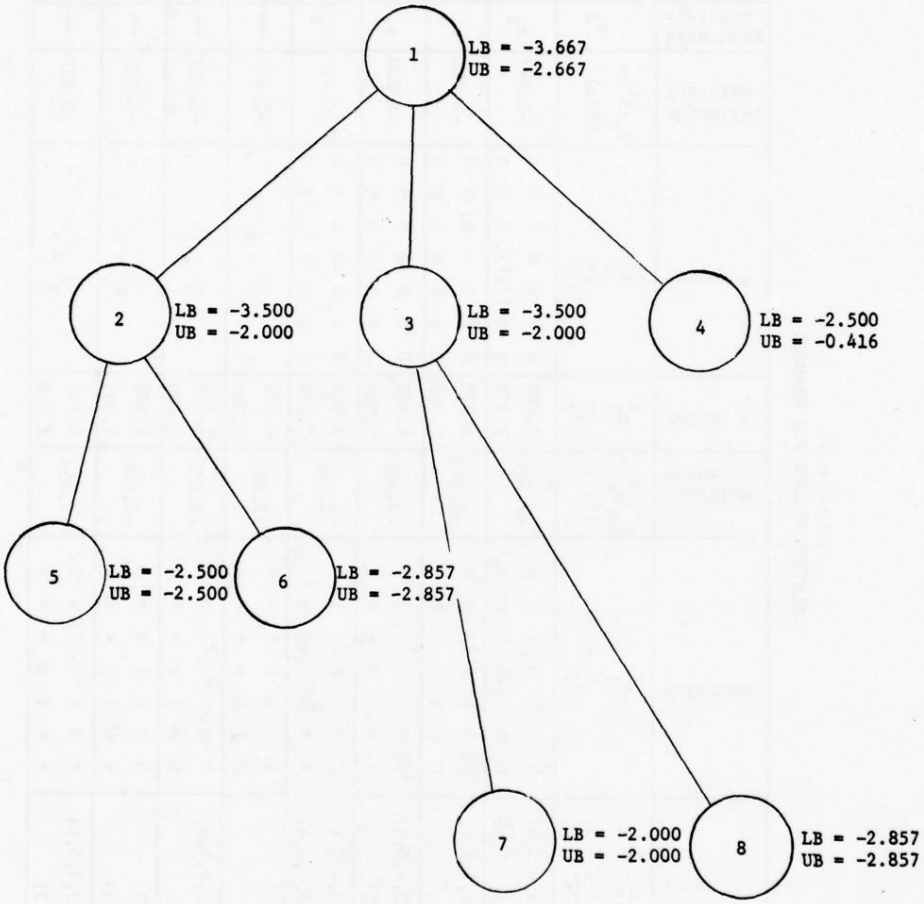


Figure 6. BRANCH AND BOUND TREE FOR EXAMPLE

A-24

5. SOME COMPUTATIONAL CONSIDERATIONS AND EXTENSIONS

In this section we point out some computational aspects of the method, some possible variations, and an extension to non-continuous problems.

We first note that each problem P^t contains m constraints corresponding to the m constraints of problem Q plus n constraints of the form $\Sigma\, \theta_{jk} = 1$. Thus the Generalized Upper Bounding Technique of Dantzig and Van Slyke [2] may be used to advantage here, and especially if n is large compared to m. This method allows one to maintain a basis of size $m \times m$.

Since each problem P^t is distinguished by the sets K_j^t, one need carry in memory only that information which identifies these sets, e.g., the first and last indices of the sets. Beale and Tomlin [1] refer to these indices as "flags". The matrix identifying the coefficients of the objective function and the first m constraints of P is common to all problems P^t. Since the basic solution of a problem being branched from is not feasible to the newly created problems, it is not clear that the basis of each problem P^t should be carried in memory along with the sets K_j^t. On the other hand, the basic solution of a problem being branched from only fails to be feasible to its descendants by virtue of one constraint and hence may be useful in creating basic feasible solutions to the newly created problems.

Once a point $\theta^{(q)}$ is found which is feasible to problem P^t, one could attempt to produce a feasible solution $\hat{\theta}^{(q)}$ which satisfies the AWR by the device outlined in Section 3. The computations necessary to produce such a point are fairly simple. If such a point $\hat{\theta}^{(q)}$ may be produced, one can immediately compute $f_o(\hat{\theta}^{(q)})$ and compare this

with the BUB(ℓ), updating this number if $f_o(\hat{\theta}^{(q)}) < BUB(\ell)$. In such a way one may be able to tighten the number BUB(ℓ) at each simplex iteration solving P^t and possibly come across an optimal solution θ^* of P during the solution of a subproblem P^t. Of course, this solution would not be recognized as such until equality occurs in (3.8).

Finally, we point out a simple modifcation of the method that will allow one to deal with piecewise continuous functions F_{ij}. In order to insure that problem Q has a solution, we assume also that each F_{ij} is lower semicontinuous. The grid points $\{y_{jk} : k \in K_j;$ $j=1,\ldots,n\}$ are chosen so that all points of discontinuity of the F_{ij}'s are among them. Let y_{JK} be a point of discontinuity of F_{LJ} and set

$$F^-_{IJ} = \lim_{x_J \uparrow y_{JK}} F_{IJ(XJ)}$$

$$F^o_{IJ} = F_{IJ}(y_{JK})$$

$$F^+_{IJ} = \lim_{x_J \downarrow y_{JK}} F_{IJ(XJ)}$$

The lower semicontinuity of F_{IJ} at y_{JK} implies that $F^o_{IJ} \leq \min \{F^-_{IJ}, F^+_{IJ}\}$. Assume, for the sake of discussion, that strict inequality holds, and define new indices K^-, K^o and K^+ corresponding to the quantities F^-_{IJ}, F^o_{IJ} and F^+_{IJ} respectively. These indices are to be ordered as

$$K - 1 < K^- < K^o < K^+ < K + 1$$

and corresponding new variables θ^-_{JK}, θ^o_{JK} and θ^+_{JK} are defined.

A-26

Problem P is thus redefined with $\theta^{-}_{JK} F^{-}_{iJ} + \theta^{o}_{JK} F^{o}_{iJ} + \theta^{+}_{JK} F^{+}_{iJ}$ replacing $\theta_{JK} F_{iJ}(y_{JK})$, $\theta^{-}_{JK} + \theta^{o}_{JK} + \theta^{+}_{JK}$ replacing θ_{JK} and $\{\ldots, K-1, K^{-}, K^{o}, K^{+}, K+1, \ldots\}$ replacing K_J.

With these modifications carried out at every point of discontinuity, the algorithm may be applied as before with no additional changes. Note that a global solution of problem P cannot have adjacent nonzero pairs $(\theta^{-}_{JK}, \theta^{o}_{JK})$ or $(\theta^{o}_{JK}, \theta^{+}_{JK})$ unless the value of $F_{oJ}(y_{JK})$ is zero, for otherwise the value of f_o could be decreased by setting $\theta^{o}_{JK} = 1$ while still maintaining feasibility. Even if $F_{oJ}(y_{JK}) = 0$ and one of the above pairs is nonzero, an equivalent feasible solution may be found for which $\theta^{o}_{JK} = 1$ and which gives the same value to $f_o(\theta)$.

In the case that $F^{o}_{IJ} = F^{-}_{IJ}$ (F_{IJ} is continuous from the left), one need only define two new variables, say θ^{o}_{JK} and θ^{+}_{JK}, and modify problem P as above. The case where F_{IJ} is right continuous is similar.

REFERENCES

[1] BEALE, E. M. L. and TOMLIN, J. A. (1970). Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables. Proceedings of the Fifth International Conference on Operations Research (J. Lawrence, ed.) 447-454. Tavistock Publications, London.

[2] DANTZIG, G. B. and VAN SLYKE, R. M. (1967). Generalized upper bounding techniques. J. Comput. System Sci. 1 213-226.

[3] FALK, J. E. and SOLAND, R. M. (1969). An algorithm for separable nonconvex programming problems. Management Sci. 15 550-569.

[4] FALK, J. E. (1969). Lagrange multipliers and nonconvex programs. SIAM J. Control 7 534-545.

[5] FALK, J. E. and HOROWITZ, J. L. (1972). Critical path problems with concave cost-time curves. Paper submitted for publication.

[6] LAWLER, E. L. and WOOD, D. E. (1966). Branch-and-bound methods: A survey. Operations Res. 14 699-719.

[7] MILLER, C. E. (1963) The simplex method for local separable programming. Recent Advances in Mathematical Programming (R. L. Graves and P. Wolfe, eds.) 89-100. McGraw Hill, New York.

[8] SOLAND, R. M. (1971). An algorithm for separable nonconvex programming problems II: Nonconvex constraints. Management Sci. 17 759-773.

[9] SOLAND, R. M. (1971). Optimal plant location with concave costs. Paper presented at 39th National Meeting of the Operations Research Society of America in Dallas, Texas.

[10] TOMLIN, J. A. (1970). Branch and bound methods for integer and non-convex programming. Integer and Nonlinear Programming (J. Abadie, ed.) 437-450. North Holland Publishing Company, Amsterdam.