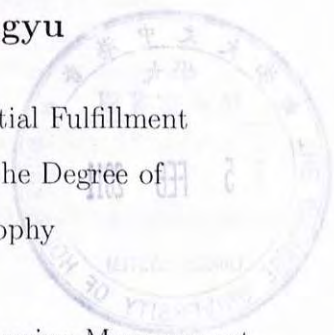


Surrogate Dual Search in Nonlinear Integer Programming

WANG, Chongyu

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Systems Engineering and Engineering Management



June 2009



Thesis/Assessment Committee

Professor Shuzhong Zhang (Chair)

Professor Duan Li (Thesis Supervisor)

Professor Janny Leung (Committee Member)

Professor Xiaoling Sun (External Examiner)

ABSTRACT

The focus of this research is on the development of efficient solution schemes for multiply constrained separable nonlinear integer programming (MCSNIP). Dynamic programming, one of the most powerful solution methodologies to achieve optimality for separable problems, suffers heavily from the notorious “curse of dimensionality”, which prevents its direct applications to MCSNIP. By aggregating multiple constraints into a single surrogate constraint, the surrogate constraint formulation offers an ideal platform for powerful utilization of dynamic programming, with a price of the existence of a duality gap in general situations. This thesis seeks a research goal to build up a framework of convergent surrogate dual search in the sense that the duality gap will be gradually eliminated during the solution process of the surrogate dual search. The overall research goal is achieved by accomplishing the following two research tasks. i) As any singly constrained separable optimization problem is corresponding to a shortest path problem and the dual value offers a lower bound of the optimal value, we propose a new formulation of the distance confined path problem and develop a solution scheme using successive network reduction. This new solution concept in turn leads to a new type of convergent surrogate dual search by removing gradually infeasible points of the primal from the feasible region of the surrogate relaxation. ii) By attaching bounds on the objective value in the surrogate constraint formulation and tightening the bounds successively using the updated dual value, the convergence to the primal optimality can be guaranteed in the surrogate dual search. Recognizing that the same function appears in both objec-

tive and constraint, we convert the doubly constrained formulation into a singly constrained one, thus facilitating effective utilization of dynamic programming. The computational results demonstrate the applicability of our proposed solution algorithms in solving large-scale MCSNIP problems. Our research extends the reach of dynamic programming to tackle successfully the long-standing challenge arisen from MCSNIP.

摘要

本论文主要研究多约束可分离非线性整数规划问题(MCSNIP)的有效求解方案。动态规划,作为最有效的求解可分离问题最优解的方法之一,由于深受为人熟知的“维数灾”的拖累,因而无法直接应用于(MCSNIP)问题。通过把多约束聚合成单一替代约束,替代约束问题为有效利用动态规划提供了一个理想的平台,而付出的代价是在一般情况下会引入对偶间隙。这篇论文的研究目的在于探讨并建立一种收敛替代对偶搜索框架,能够逐步消除对偶间隙以至找出原问题的最优解。我们通过完成以下两件任务来达到整体的研究目标:1)由于任意单约束可分离最优化问题对应于一个最短路问题,且对偶值提供最优解的下界,我们提出一种新的距离限制路径问题模型并开拓了一种网络逐步简化的求解方案。这种新的求解思想进一步引出一种新的收敛对偶替代搜索,逐步从替代松弛可行域中移除原问题的不可行点。2)在替代约束问题中添加目标值的上下界限并应用更新的对偶值不断收紧目标值上下界范围,我们能保证对偶搜索收敛至原问题最优解。注意到目标函数和一个约束函数相同,我们把双约束的问题转化成单约束问题,从而便利使用动态规划。计算结果显示了我们提出的算法在解决大规模 MCSNIP问题的适用性。我们的研究拓展了动态规划的应用范围,特别能成功应对来自 MCSNIP问题造成的长期挑战。

ACKNOWLEDGEMENT

I would like to express my greatest gratitude to my supervisor, Professor Duan Li, for his invaluable advices and excellent guidance in the development of the ideas in this thesis and all the help throughout my postgraduate studies.

I am also thankful to Professor Xiaoling Sun for serving as the external examiner of my thesis and grateful to Professor Shuzhong Zhang and Professor Janny Leung for their expert serving as members of my thesis committee. Moreover, I would like to thank Professor Yirong Yao, Dr. Jun Wang and Dr. Chi-Kong Ng for their professional suggestions in the course of this work.

I also want to thank my fellow members, Chuili Liu, Jianjun Gao, Lan Yi, Xiangyu Cui, Shengsheng Gu and Xiaojin Zheng for their enthusiastic helps and valuable discussions.

Many thanks to my good friend Xiangwei Wan, roommate Di Wu and all my friends in Chinese University of Hong Kong. I really enjoy the happy times spent with them.

Finally, special thanks should be devoted to the girl who changed my life.

CONTENTS

Abstract	1
Abstract in Chinese	3
Acknowledgement	4
Contents	5
List of Tables	7
List of Figures	8
1. Introduction	9
2. Conventional Dynamic Programming	15
2.1. Principle of optimality and decomposition	15
2.2. Backward dynamic programming	17
2.3. Forward dynamic programming	20
2.4. Curse of dimensionality	23
3. Surrogate Constraint Formulation	26
3.1. Surrogate constraint formulation	26
3.2. Singly constrained dynamic programming	28
3.3. Surrogate dual search	29

4. Distance Confined Path Algorithm	34
4.1. Yen's algorithm for the kth shortest path problem	35
4.2. Application of Yen's method to integer programming	36
4.3. Distance confined path problem	42
4.4. Application of distance confined path formulation to integer programming	50
5. Convergent Surrogate Dual Search	59
5.1. Algorithm for convergent surrogate dual search	62
5.2. Solution schemes for $(P_\mu(\alpha^k, \beta))$ and $f(x) = \alpha^k$	63
5.3. Computational Results and Analysis	68
6. Conclusions	72
Bibliography	74

LIST OF TABLES

2.1. Solution process for Example 2.2.1 using backward dynamic programming	19
2.2. Solution process for Example 2.3.1 using forward dynamic programming	22
4.1. The extended dynamic programming table in Example 4.2.1	43
4.2. The value table in Example 4.3.1	48
4.3. The updated value table in Example 4.3.1	49
4.4. The original value table of $F_{\mu 1}$ in Example 4.2.1	52
4.5. The updated value table of $F_{\mu 1}$ in Example 4.2.1	54
4.6. The original value table of $F_{\mu 3}$ in Example 4.2.1	55
4.7. The updated value table of $F_{\mu 3}$ in Example 4.2.1	56
4.8. The updated value table of P_{μ} in Example 4.2.1	57
5.1. Solution process for problem D using forward dynamic programming	66
5.2. Solution process for problem D using forward dynamic programming incorporated with surrogate constraint.	67
5.3. Iteration dynamic programming process in Example 5.3.1	68
5.4. Numerical results for third degree polynomial integer programming	69
5.5. Comparison with Lagrangian dual search method($n = 30, m = 20$)	70
5.6. Comparison with Lagrangian dual search method	70

LIST OF FIGURES

4.1. Network in Example 4.2.1	38
4.2. Shortest path in Example 4.2.1	39
4.3. Deviating paths in Example 4.2.1	40
4.4. Graph in Example 4.3.1	47
4.5. Graph in Example 4.3.1	49
4.6. Graphical presentation of $F_{\mu 1}$	53
4.7. Graphical presentation of reduced version 1 of $F_{\mu 1}$	54
4.8. Graphical presentation of reduced version 2 of $F_{\mu 3}$	55
4.9. Graphical presentation of reduced version 3 of $F_{\mu 3}$	56
4.10. Graphical presentation of reduced version 4 of P_{μ}	57
4.11. Graphical presentation of reduced version 5 of P_{μ}	58

CHAPTER 1

INTRODUCTION

We consider in this research the following general class of multiply constrained separable integer programming problems,

$$(P) \quad \min f(x) = \sum_{j=1}^n f_j(x_j)$$
$$\text{s.t. } g_i(x) = \sum_{j=1}^n g_{ij}(x_j) \leq b_i, \quad i = 1, \dots, m,$$
$$x \in X = X_1 \times X_2 \times \dots \times X_n,$$

where all f_j 's and g_{ij} 's are real-valued functions defined on R , and all X_j 's are finite integer sets in R . Problem (P) covers very general situations of nonlinear integer programming problems as no additional property such as linearity, convexity, concavity, monotonicity or differentiability is assumed in (P) . Problem (P) possesses a nonconvex nature in many instances, e.g., concave integer programming [2][4][17] and polynomial integer programming [18].

Problem (P) has a wide variety of applications, including resource allocation problems and nonlinear multi-dimensional knapsack problems. In particular, capital budgeting, manufacturing capacity planning, production planning, network reliability, stratified sampling are special cases of (P) .

Integer programming has been one of the great challenges in front of the optimization research community for many years, due to an exponential growth in its computational complexity with respect to the problem dimension. It has been

shown in the literature that many special cases of (P) are NP-hard [8][19][38]. Therefore, constructing an efficient exact algorithm for (P) is a challenging task.

The literature on the solution methods of (P) has been dominated by the results for singly constrained situations until recently. Ibaraki and Katoh [19] summarized certain algorithms for singly constrained resource allocation problems where the objective function is convex and separable and the single constraint is of a special form of $\sum_{j=1}^n x_j = N$. Bretthauer and Shetty [5] proposed a branch-and-bound algorithm for a special singly constrained case of (P) where all f_j 's and g_{ij} 's are convex. Hochbaum [16] studied a singly constrained case of (P) where all f_j 's and g_{ij} 's are convex and monotonically nonincreasing. The piecewise linear approximations of f_j 's and g_{ij} 's are used in [16] to convert the problem into a 0-1 linear integer programming problem.

The concept of duality plays an important role in discrete optimization. The Lagrangian relaxation methods are widely adopted in integer programming (see, e.g., [11][12][13][39]). As discussed in [33], the conventional Lagrangian dual method often fails to generate an optimal solution to (P) due to the existence of a duality gap. Using group theory, Bell and Shapiro [1] proposed a convergent Lagrangian duality theory for linear integer programming in which the duality gap is reduced by reshaping the feasible region. Recently, the duality gap in general nonlinear integer programming was examined and its related properties were studied in [28][33]. Nonlinear Lagrangian formulations are proposed in [28][33][40] to offer a success guarantee for the dual search in generating an optimal solution of the primal integer programming problem. Although the nonlinear Lagrangian formulations possess strong duality or asymptotic strong duality, it does not lead to a decomposability which is crucial for an efficient implementation of a dual scheme.

Along with the Lagrangian duality theory, the surrogate duality theory has been widely used in solving integer programming problems. While the Lagrangian dual formulation generates a relaxation by incorporating the constraints into the objective function, the surrogate dual generates a relaxation by aggre-

gating multiple constraints into a single surrogate constraint. To eliminate the duality gap, Li [27] proposes a nonlinear surrogate dual method which guarantees the equivalence between the primal problem and its relaxation and eliminates the need of dual search. However, the resulting nonlinear surrogate constraint formulation is, in general, more difficult to solve than the primal problem, as the separability of the primal problem (P) is destroyed.

The past few years have witnessed research efforts in developing implementable solution schemes to identify the exact solution of (P) in a process of gradually reducing duality gap via an integration of Lagrangian dual search and various cutting schemes.

Li, Wang and Sun [32] develop a convergent Lagrangian method for (P) using objective cuts. The algorithm starts with a lower bound derived from the dual value by the conventional Lagrangian dual search and an upper bound by a feasible solution generated in the dual search (if any). The lower level cut and upper level cut are imposed to (P) such that the duality bound (duality gap) is forced to shrink. The objective cut is updated successively with the range between the upper cut and the lower cut monotonically decreasing. The algorithm terminates in a finite number of iterations, either reaching an optimal solution to (P) or reporting an infeasibility of (P).

For problem (P) with a quadratic objective function, Li, Sun and Wang [29] propose a solution method that combines the Lagrangian dual method with a duality reduction scheme using contour-cut. At each iteration of the algorithm, lower and upper bounds of the problem are determined by the Lagrangian dual search. To eliminate the duality gap, a cut-and-partition scheme is derived by exploring the special structure of the quadratic contour. The method finds an exact solution of the problem in a finite number of iterations.

For the nonlinear multi-dimensional knapsack problem, a special case of (P), Li, Sun, Wang and McKinnon [31] develop a convergent Lagrangian and domain cut method. The proposed method exploits the special structure of the problem by Lagrangian decomposition and dual search. The domain cut is used

to eliminate the duality gap and thus to guarantee the finding of an optimal exact solution to the primal problem.

Dynamic programming pioneered by Richard Bellman in 1950's is one of the most powerful methodologies for separable optimization problems. However, it suffers heavily from the notorious "curse of dimensionality" which prevents a direct application when a large number of constraints are present. Mitigating the curse of dimensionality in dynamic programming has been a challenging research task in front of the control and optimization community for many years. A few solution algorithms have been suggested in the literature for alleviating the "curse of dimensionality" in dynamic programming.

Recognizing a relationship between the optimal solutions and the efficient solutions in the constraint space, a hybrid method was developed in [36] with a purpose to fathom in the solution process inefficient incomplete feasible solutions by bounds and dominance rules.

Many attempts have been made to mitigate the curse of dimensionality of dynamic programming in its control applications.

A successive approximation technique was proposed in [24][25] for a discrete-time deterministic optimal control problem. A nominal trajectory of state x and control u are specified first. One of the n state variables is selected each time to be optimized while the others are held fixed. The procedure repeats such that each of the state variables is selected at least once. Thus, the original n -dimensional problem is transformed to a sequence of one dimensional problems which can be effectively handled by dynamic programming. However, the convergence to the global solution is not guaranteed and this method may be trapped in a local minimum.

Differential dynamic programming (DDP) developed in [20][34][37][41] is a second-order method that successively improves the incumbent trajectory under a convexity assumption based on the principle of optimality. The advantage of DDP over traditional dynamic programming is that it does not require discretization of the state space, thus avoiding the "cures of dimensionality". However,

convergence issues may arise and paper [34] addresses the convergence issues of differential dynamic programming (DDP).

The idea of region reduction was adopted in [35] by successively refining a coarse grid assignment of the state space.

Note that the curse of dimensionality disappears when an analytical form of the cost-to-go function can be achieved. Thus, different numerical methods, such as linear and spline interpolation [21] and neural computing [3], have been suggested in the literature to approximate the cost-to-go by an analytical form.

The focus of this research is on the development of efficient solution schemes for problem (P). By aggregating multiple constraints into a single surrogate constraint, the surrogate constraint formulation offers an ideal platform for powerful utilization of dynamic programming, albeit with a price of the existence of a duality gap in general situations. This research seeks a research goal to build up a framework of convergent surrogate dual search in the sense that the duality gap will be gradually eliminated during the solution process of the surrogate dual search. The overall research goal is achieved by accomplishing the following two research tasks.

i) As any singly constrained separable optimization is corresponding to a shortest path problem and the dual value offers a lower bound of the optimal value, we propose a new formulation of the distance confined path problem and develop a solution scheme using successive network reduction. This new solution concept in turn leads to a new type of convergent surrogate dual search by removing gradually infeasible points of the primal problem from the feasible region of the surrogate relaxation.

ii) By attaching bounds on the objective value in the surrogate constraint formulation and tightening the bounds successively using the updated dual value, the convergence to the primal optimality can be guaranteed in the surrogate dual search. Recognizing the same function appears in both objective and constraint, we convert the doubly constrained formulation into a singly constrained one, thus facilitating effective utilization of dynamic programming. The computa-

tional results demonstrate the applicability of our proposed solution algorithms in solving large-scale instances of (P) .

The structure of this dissertation is as follows. After the introduction given in this chapter, we review dynamic programming and the surrogate constraint formulation, respectively, in Chapters 2 and 3. We propose a new formulation of distance defined path problem and develop a solution algorithm in Chapter 4 and discuss its applications in networks. This new problem formulation and its corresponding graphical solution algorithm lead further a successive reduction scheme to reduce infeasible and/or non-optimal points from the dynamic table of the surrogate constraint formulation for (P) , resulting in our first convergent surrogate dual search algorithm. In Chapter 5, we consider our second convergent surrogate dual search algorithm for problem (P) by attaching a bound constraint on the objective value in the surrogate constraint formulation. By successively reducing the range of the bounds, we ensure a convergence to the solution of the primal problem (P) under this solution algorithm. Recognizing the special structure of the objective confined surrogate constraint formulation, we convert the doubly constrained formulation into its equivalent singly constrained counterpart, thus facilitating effective utilization of dynamic programming. We further demonstrate the efficiency of the proposed algorithm in numerical tests. We finally summarize this research in Chapter 6. Our research extends the reach of dynamic programming to tackle successfully the long-standing challenge arisen from problem (P) .

□ End of chapter.

CHAPTER 2

CONVENTIONAL DYNAMIC PROGRAMMING

Dynamic programming has been widely adopted as a solution scheme for discrete optimization. The separability of both the objective function f and constraint function g_i 's in (P) makes dynamic programming method an ideal technique. The following assumption is essential for an efficient implementation of a dynamic programming method for (P) .

Assumption 2.0.1. *Function g_{ij} is integer-valued, for all $j = 1, \dots, n$ and $i = 1, \dots, m$.*

2.1. Principle of optimality and decomposition

To apply dynamic programming in solving the problem (P) , we need to introduce the stage variable k and state vector $s_k \in \mathbf{R}^m$ at stage k that satisfies the following recursion:

$$s_{k+1} = s_k + g^k(x_k), \quad k = 1, \dots, n,$$

with an initial condition $s_1 = 0$, where

$$g^k(x_k) = (g_{1k}(x_k), \dots, g_{mk}(x_k))^T.$$

Since the constraints are integer-valued, we only need to consider integer points in the state space. Furthermore, the feasible region of the state vector at

stage k with $2 \leq k \leq n + 1$ can be confined as follows:

$$\underline{s}_k \leq s_k \leq \bar{s}_k,$$

where

$$\underline{s}_k = \begin{pmatrix} \sum_{t=1}^{k-1} \min_{x_t \in X_t} g_{1t}(x_t) \\ \vdots \\ \sum_{t=1}^{k-1} \min_{x_t \in X_t} g_{mt}(x_t) \end{pmatrix} \quad (2.1.1)$$

and

$$\bar{s}_k = \begin{pmatrix} \min\{\sum_{t=1}^{k-1} \max_{x_t \in X_t} g_{1t}(x_t), b_1 - \sum_{t=k}^n \min_{x_t \in X_t} g_{1t}(x_t)\} \\ \vdots \\ \min\{\sum_{t=1}^{k-1} \max_{x_t \in X_t} g_{mt}(x_t), b_m - \sum_{t=k}^n \min_{x_t \in X_t} g_{mt}(x_t)\} \end{pmatrix} \quad (2.1.2)$$

The principle of optimality revealed by Richard Bellman in his pioneering work in 1950's is the cornerstone of dynamic programming. The following is a backward version of principle of optimality originally stated by Bellman in his seminal work:

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Similarly, a forward version of the principle of optimality can be stated as follows:

An optimal policy has the property that whatever the later states and later decisions will be, the early portion of the decisions must constitute an optimal policy with regard to the intermediate state from which the later portion of the decisions starts to apply.

The principle of optimality enables us to decompose the primal n -variable optimization problem into a family of univariate optimization problems, thus reducing significantly the computational efforts. Dynamic programming can be applied to solve problem (P) either by a backward recursion or by a forward recursion.

2.2. Backward dynamic programming

For a given state s at stage k , $1 \leq k \leq n$, define the cost-to-go function as follows,

$$\begin{aligned} \hat{t}_k(s) &= \min \sum_{j=k}^n f_j(x_j), \\ \text{s.t. } s + \sum_{j=k}^n g_j(x_j) &\leq b, \\ x_j &\in X_j, \quad j = k, \dots, n. \end{aligned} \quad (2.2.1)$$

It is obvious that

$$v(P) = \hat{t}_1(0).$$

Based on Bellman's principle of optimality, the cost-to-go function satisfies the following backward recursive relation for $k = n - 1, n - 2, \dots, 1$,

$$\hat{t}_k(s) = \min_{x_k \in X_k} \{f_k(x_k) + \hat{t}_{k+1}(s + g^k(x_k))\}$$

with boundary condition

$$\hat{t}_n(s) = \min_{x_n \in X_n} \{f_n(x_n) | s + g^n(x_n) \leq b\}.$$

Define

$$x_n^*(s) = \arg \min_{x_n \in X_n} \{f_n(x_n) | s + g^n(x_n) \leq b\},$$

and

$$x_k^*(s) = \arg \min_{x_k \in X_k} \{f_k(x_k) + \hat{t}_{k+1}(s + g^k(x_k))\}, \quad k = n - 1, \dots, 1.$$

The backward dynamic programming starts at $k = n$ and moves backwards, $k = n - 1, \dots, 1$. It calculates the cost-to-go recursively for every s between \underline{s}_k and \bar{s}_k at stage k and finally stops at $s_1 = 0$. The tracing process is then carried

out in a forward way to identify the optimal solution of (P) . Starting from $x_1^*(0)$, the optimal state at stage 2 is obtained as $s_2^* = g^1(x_1^*(0))$. The algorithm then identifies the optimal solution at stage 2, $x_2^*(s_2^*)$, which yields the optimal state at stage 3, $s_3^* = s_2^* + g^2(x_2^*(s_2^*))$. The process terminates when it reaches s_n^* and finds out $x_n^*(s_n^*)$.

Example 2.2.1.

$$\begin{aligned} \min f(x) &= 2x_1^2 - 3x_2^3 + 5x_3 \\ \text{s.t. } g_1(x) &= -2x_1 + 2x_2 + x_3^2 \leq 1, \\ g_2(x) &= x_1^3 + x_2 - x_3^2 \leq 0, \\ x_i &\in \{-1, 0, 1\}, i = 1, 2, 3. \end{aligned}$$

It can be checked that the optimal solution of this small-scale example is $x^* = (0, 0, -1)^T$ with $f(x^*) = -5$. We now illustrate how to identify the optimal solution by using dynamic programming.

Using formulas in (2.1.1) and (2.1.2), the feasible regions of the state vector can be found as follows for $k = 2, 3$ and 4,

$$\begin{aligned} \begin{pmatrix} -2 \\ -1 \end{pmatrix} &\leq s_2 \leq \begin{pmatrix} 2 \\ 1 \end{pmatrix} \\ \begin{pmatrix} -4 \\ -2 \end{pmatrix} &\leq s_3 \leq \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ \begin{pmatrix} -4 \\ -3 \end{pmatrix} &\leq s_4 \leq \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{aligned}$$

Table 2.1 gives the solution processes using backward dynamic programming.

The solution process using backward dynamic programming starts from stage 3. For each possible s_3 , the optimal decision $x_3^*(s_3)$ is found and the corresponding optimal cost-to-go $\hat{t}_3(s_3)$ is recorded. For example, at $s_3 = (1, -2)^T$, both $x_3 = 1$ and $x_3 = -1$ are infeasible. The optimal decision $x_3^*((1, -2)^T)$ is

Table 2.1: Solution process for Example 2.2.1 using backward dynamic programming

s_1	$x_1^*(s_1)/\hat{t}_1(s_1)$	s_2	$x_2^*(s_2)/\hat{t}_2(s_2)$	s_3	$x_3^*(s_3)/\hat{t}_3(s_3)$
$(0, 0)^T$	$0 / -5^*$	$(-2, -1)^T$	$1 / -8$	$(-4, -2)^T$	$-1 / -5$
		$(-2, 0)^T$	$1 / -8$	$(-4, -1)^T$	$-1 / -5$
		$(-2, 1)^T$	$0 / -5$	$(-4, 0)^T$	$-1 / -5$
		$(-1, -1)^T$	$0 / -5$	$(-4, 1)^T$	$-1 / -5$
		$(-1, 0)^T$	$0 / -5$	$(-3, -2)^T$	$-1 / -5$
		$(-1, 1)^T$	$0 / -5$	$(-3, -1)^T$	$-1 / -5$
		$(0, -1)^T$	$0 / -5$	$(-3, 0)^T$	$-1 / -5$
		$(0, 0)^T$	$0 / -5$	$(-3, 1)^T$	$-1 / -5$
		$(0, 1)^T$	$0 / -5$	$(-2, -2)^T$	$-1 / -5$
		$(1, -1)^T$	$0 / 0$	$(-2, -1)^T$	$-1 / -5$
		$(1, 0)^T$	$0 / 0$	$(-2, 0)^T$	$-1 / -5$
		$(1, 1)^T$	$-1 / -2$	$(-2, 1)^T$	$-1 / -5$
		$(2, -1)^T$	$-1 / -2$	$(-1, -2)^T$	$-1 / -5$
		$(2, 0)^T$	$-1 / -2$	$(-1, -1)^T$	$-1 / -5$
		$(2, 1)^T$	$-1 / -2$	$(-1, 0)^T$	$-1 / -5$
				$(-1, 1)^T$	$-1 / -5$
				$(0, -2)^T$	$-1 / -5$
				$(0, -1)^T$	$-1 / -5$
				$(0, 0)^T$	$-1 / -5$
				$(0, 1)^T$	$-1 / -5$
				$(1, -2)^T$	$0 / 0$
				$(1, -1)^T$	$0 / 0$
				$(1, 0)^T$	$0 / 0$
				$(1, 1)^T$	inf / ∞

found to be 0 and the corresponding $\hat{t}_3((1, -2)^T)$ is 0. If there does not exist a feasible solution at s_3 , $x_3^*(s_3)$ is set as ∞ . Then, we move back to stage 2. At each possible s_2 , we compare $f_2(x_2) + \hat{t}_3(s_2 + g^2(x_2))$ for $x_2 = -1, 0$ and 1 and find out $x_2^*(s_2)$ and the corresponding optimal cost-to-go $\hat{t}_2(s_2)$. For example, at $s_2 = (-1, -1)^T$, comparison of $-3(-1) + \hat{t}_3((1, 0)^T) = 3$, $-3(0) + \hat{t}_3((-1, 1)^T) = -5$ and $-3(1)^3 + \hat{t}_3((-3, -2)^T) = -2$ yields $x_2^*((-1, -1)^T) = 0$ and $\hat{t}_2((-1, -1)^T) = -5$. Finally, we move back to stage 1. Checking $f_1(x_1) + \hat{t}_2((0, 0)^T + g^1(x_1))$ for $x_1 = -1, 0$ and 1 gives $x_1^*(s_1 = (0, 0)^T) = 0$ and $\hat{t}_1(s_1 = (0, 0)^T) = -5$. Tracing back, we find the optimal solution for the example problem: $x_1 = x_2 = 0$ and $x_3 = -1$.

2.3. Forward dynamic programming

For a given state s at stage k , $2 \leq k \leq n + 1$, define the cost-to-accumulate function as follows,

$$\begin{aligned} \tilde{t}_k(s) &= \min \sum_{j=1}^{k-1} f_j(x_j), \\ \text{s.t. } &\sum_{j=1}^{k-1} g^j(x_j) \leq s, \\ &x_j \in X_j, j = 1, \dots, k-1. \end{aligned} \quad (2.3.1)$$

It is obvious that

$$v(P) = \min \{ \tilde{t}_{n+1}(s) | s \leq b \}.$$

Based on the forward version of Bellman's optimality principle, the cost-to-accumulate function satisfies the following forward recursive relation for $k = 1, \dots, n + 1$,

$$\tilde{t}_k(s) = \min_{x_{k-1} \in X_{k-1}} \{ f_{k-1}(x_{k-1}) + \tilde{t}_{k-1}(s - g^{k-1}(x_{k-1})) \},$$

with boundary condition

$$\tilde{t}_2(s) = \min_{x_1 \in X_1} \{ f_1(x_1) | g^1(x_1) \leq s \},$$

Define

$$x_1^*(s) = \arg \min_{x_1 \in X_1} \{f_1(x_1) | g^1(x_1) \leq s\},$$

and

$$x_{k-1}^*(s) = \arg \min_{x_{k-1} \in X_{k-1}} \{f_{k-1}(x_{k-1}) + \tilde{t}_{k-1}(s - g^{k-1}(x_{k-1}))\},$$

$$k = 2, \dots, n + 1.$$

The forward dynamic programming starts at $k = 2$ and moves forward, $k = 3, \dots, n + 1$. It calculates the cost-to-accumulate recursively for every s at stage k between \underline{s}_k and \bar{s}_k and finally stops at stage $n + 1$. Let

$$s_{n+1}^* = \arg \min \{\tilde{t}_{n+1}(s) | s \leq b\}.$$

The tracing process is then carried out in a backward way to identify the optimal solution of P . Starting from $x_n^*(s_{n+1}^*)$, the optimal state at stage n is obtained as $s_n^* = s_{n+1}^* - g^n(x_n^*(s_{n+1}^*))$. The algorithm then identifies the optimal solution at stage n , $x_{n-1}^*(s_n^*)$, which yields the optimal state at stage $n - 1$, $s_{n-1}^* = s_n^* - g^{n-1}(x_{n-1}^*(s_n^*))$. The process terminates when it reaches s_2^* and finds out $x_1^*(s_2^*)$.

Example 2.3.1. *We solve again Example 2.2.1, but this time, by forward dynamic programming. The forward dynamic programming starts from stage 2 and ends at stage 4. Minimizing \tilde{t}_4 with respect to $s_4 \leq (1, 0)^T$ finds out the optimal value of the example problem $\tilde{t}_4((1, -1)^T) = -5$. Tracing back identifies optimal solution: $x_3^* = -1, x_2^* = 0, x_1^* = 0$. Table 2.2 presents the solution process using forward dynamic programming.*

Determining the feasible region could become a tedious task in applying dynamic programming. This difficulty can be alleviated to certain degree when the following approach is adapted[30].

Table 2.2: Solution process for Example 2.3.1 using forward dynamic programming

s_2	$x_1^*(s_2)/\hat{t}_2(s_2)$	s_3	$x_2^*(s_3)/\hat{t}_3(s_3)$	s_4	$x_3^*(s_4)/\hat{t}_4(s_4)$
$(-2, -1)^T$	inf/ ∞	$(-4, -2)^T$	inf/ ∞	$(-4, -3)^T$	inf/ ∞
$(-2, 0)^T$	inf/ ∞	$(-4, -1)^T$	inf/ ∞	$(-4, -2)^T$	inf/ ∞
$(-2, 1)^T$	1/2	$(-4, 0)^T$	-1/5	$(-4, -1)^T$	inf/ ∞
$(-1, -1)^T$	inf/ ∞	$(-4, 1)^T$	inf/ ∞	$(-4, 0)^T$	0/5
$(-1, 0)^T$	inf/ ∞	$(-3, -2)^T$	inf/ ∞	$(-3, -3)^T$	inf/ ∞
$(-1, 1)^T$	inf/ ∞	$(-3, -1)^T$	inf/ ∞	$(-3, -2)^T$	inf/ ∞
$(0, -1)^T$	inf/ ∞	$(-3, 0)^T$	inf/ ∞	$(-3, -1)^T$	-1/0
$(0, 0)^T$	0/0	$(-3, 1)^T$	inf/ ∞	$(-3, 0)^T$	inf/ ∞
$(0, 1)^T$	0/0	$(-2, -2)^T$	inf/ ∞	$(-2, -3)^T$	inf/ ∞
$(1, -1)^T$	inf/ ∞	$(-2, -1)^T$	-1/3	$(-2, -2)^T$	inf/ ∞
$(1, 0)^T$	0/0	$(-2, 0)^T$	-1/3	$(-2, -1)^T$	0/3
$(1, 1)^T$	0/0	$(-2, 1)^T$	0/2	$(-2, 0)^T$	inf/ ∞
$(2, -1)^T$	-1/2	$(-1, -2)^T$	inf/ ∞	$(-1, -3)^T$	inf/ ∞
$(2, 0)^T$	0/0	$(-1, -1)^T$	-1/3	$(-1, -2)^T$	-1/-2
$(2, 1)^T$	0/0	$(-1, 0)^T$	-1/3	$(-1, -1)^T$	inf/ ∞
		$(-1, 1)^T$	inf/ ∞	$(-1, 0)^T$	-1/-3
		$(0, -2)^T$	-1/5	$(0, -3)^T$	inf/ ∞
		$(0, -1)^T$	inf/ ∞	$(0, -2)^T$	0/5
		$(0, 0)^T$	0/0	$(0, -1)^T$	inf/ ∞
		$(0, 1)^T$	inf/ ∞	$(0, 0)^T$	0/0
		$(1, -2)^T$	inf/ ∞	$(1, -3)^T$	-1/0
		$(1, -1)^T$	inf/ ∞	$(1, -2)^T$	inf/ ∞
		$(1, 0)^T$	inf/ ∞	$(1, -1)^T$	-1/-5*
		$(1, 1)^T$	inf/ ∞	$(1, 0)^T$	inf/ ∞

Assumption 2.3.1. For all $j = 1, \dots, n$ and $i = 1, \dots, m$, function g_{ij} is integer valued and is nonnegative for all $x_j \in X_j$.

When Assumption 2.3.1 is satisfied, the range of s_k at stage k , for $k = 2, \dots, n + 1$, can be simply determined by $[(0, \dots, 0)^T, (b_1, \dots, b_m)^T]$.

If the nonnegativity assumption does not hold for some g_{ij} , then we can subtract $\min_{x_j \in X_j} g_{ij}(x_j)$ from both g_{ij} and b_i at the same time. Repeating this equivalent transformation for all g_{ij} 's that do not possess the nonnegativity property such that Assumption 2.3.1 holds for the transformed problem. The range of $(s_k)_i$ at stage k for $k = 2, \dots, n + 1$ can be then given by $[0, b_i - \sum_{j \in I_i} \min_{x_j \in X_j} g_{ij}]$, where $I_i = \{j = 1, \dots, n \mid \min_{x_j \in X_j} g_{ij} < 0\}$. The price to perform such a transformation is an enlargement of the feasible region of the state space which affects an efficient implementation of dynamic programming.

It is evident that the number of the possible states increases exponentially with respect to the number of constraints. Thus, although dynamic programming is conceptually an ideal solution scheme for separable integer programming, the "curse of dimensionality" prevents its direct application to multiple constrained cases of (P) when m is large. Dynamic programming, however, remains as an efficient solution scheme for separable integer programming problem when m is small, especially for singly constrained cases.

2.4. Curse of dimensionality

Consider the following problem with 3 variables and 5 constraints:

$$\begin{aligned}
\min f(x) &= 2x_1^2 - 3x_2^3 + 5x_3 \\
\text{s.t. } g_1(x) &= -2x_1 + 2x_2 + x_3^2 \leq 1, \\
g_2(x) &= x_1^3 + x_2 - x_3^2 \leq 0, \\
g_3(x) &= -x_1^2 - 2x_2 + 2x_3^3 \leq 0, \\
g_4(x) &= -x_1^2 - x_2^2 - x_3^2 \leq -1, \\
g_5(x) &= x_1^3 - 2x_2 - 3x_3 \leq 3, \\
x_i &\in \{-1, 0, 1\}, i = 1, 2, \dots, 5.
\end{aligned}$$

By the formula in 2.1.1 and 2.1.2, we get the feasible region of the state vector as follows at stage 2 and 3,

$$\begin{aligned}
\begin{pmatrix} -2 \\ -1 \\ -1 \\ -1 \\ -1 \end{pmatrix} \leq s_2 \leq \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\
\begin{pmatrix} -4 \\ -2 \\ -3 \\ -2 \\ -3 \end{pmatrix} \leq s_3 \leq \begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \\ 3 \end{pmatrix}
\end{aligned}$$

The numbers of the states in stage 2 and stage 3 are $5 * 3 * 2 * 2 * 3 = 180$ and $6 * 4 * 6 * 3 * 7 = 3024$ respectively. From the calculation, we see that the number of states in its feasible region does increase exponentially with respect to the number of constraints. Therefore, dynamic programming method becomes inefficient when m is large as it requires huge computational efforts and storage spaces. Besides, it will also cost much more computation efforts if the state range is comparatively large. This phenomenon which prevents us using dynamic

programming to solve the problem (P) directly is termed by Richard Bellman as “curse of dimensionality”.

It is unfortunate that the number of states increases exponentially with the number of constraints m when adopting dynamic programming method. The term “curse of dimensionality” describes the dimensionality problem caused by the exponential increase in the state space, resulting in a significant obstacle in solving large-scale instances of (P) by numerical backwards induction.

□ End of chapter.

CHAPTER 3

SURROGATE CONSTRAINT FORMULATION

The surrogate duality theory has been developed in solving mathematical programming problems, including both continuous optimization and integer programming problems. While the dynamic programming method is inefficient to solve directly multiply constrained separable integer programming problems, the surrogate constraint formulation generates a platform for an efficient utilization of dynamic programming by aggregating multiple constraints into a single surrogate constraint.

3.1. Surrogate constraint formulation

Let $g(x) = (g_1(x), \dots, g_m(x))^T$ and $b = (b_1, \dots, b_m)^T$. Aggregating the multiple major constraints of (P) into a single surrogate constraint yields the following surrogate constrained formulation,

$$\begin{aligned} (P_\mu) \quad & \min f(x) \\ & \text{s.t. } \mu^T(g(x) - b) \leq 0 \\ & x \in X, \end{aligned}$$

where $\mu = (\mu_1, \dots, \mu_m)^T \in \mathbf{R}_+^m$ is a vector of surrogate multipliers. Define S to

be the feasible region of decision vectors in (P) ,

$$S = \{x \in X | g_i(x) \leq b_i, i = 1, 2, \dots, m\},$$

and $S(\mu)$ to be the feasible region of decision vectors in $P(\mu)$,

$$S(\mu) = \{x \in X | \mu^T(g(x) - b) \leq 0\}. \quad (3.1.1)$$

Since $S \subseteq S(\mu)$, $\forall \mu \in \mathbf{R}_+^m$, (P_μ) is a relaxation of (P) .

Denote by $v(Q)$ the optimal value of an optimization problem (Q) . The surrogate dual is an optimization problem in μ ,

$$\begin{aligned} (D_s) \quad & \max v(P_\mu) \\ & \text{s.t. } \mu \in \mathbf{R}_+^m. \end{aligned}$$

The following weak surrogate duality is evident,

$$v(P_\mu) \leq v(P), \quad \forall \mu \in \mathbf{R}_+^m.$$

Consequently, the surrogate dual provides a lower bound for $v(P)$.

$$v(D_s) \leq v(P).$$

Since $S \subseteq S(\mu)$, $\forall \mu \in \mathbf{R}_+^m$, a minimizer, x^* , over $S(\mu^*)$ with $\mu^* \in \mathbf{R}_+^m$ and $x^* \in S$ must be also a minimizer over S . Furthermore, from the weak surrogate duality and from the fact that problems (P) and (P_μ) have the same objective function, we have $f(x^*) = v(P_{\mu^*}) \leq v(D_s) \leq v(P) = f(x^*)$. Therefore, $v(D_s) = v(P)$. In summary, we have the following strong surrogate duality theorem.

Theorem 3.1.1. [30](STRONG SURROGATE DUALITY) *If an x^* solves P_{μ^*} for a $\mu^* \in \mathbf{R}_+^m$ and x^* is feasible in (P) , then x^* solves (P) and $v(D_s) = v(P)$.*

It is clear that $v(D_s) = v(P_{\theta\mu})$ for any $\theta > 0$. Thus, the surrogate dual problem (D_s) can be normalized to an equivalent problem with a compact feasible region:

$$\begin{aligned} (D_s^n) \quad & \max v(P_\mu) \\ & \text{s.t. } \mu \in \Lambda, \end{aligned}$$

where $\Lambda = \{\mu \in \mathbf{R}_+^m | e^T \mu \leq 1\}$ and $e = (1, \dots, 1)^T$.

3.2. Singly constrained dynamic programming

Problem (P_μ) is a singly constrained separable integer programming problem and can be solved efficiently by dynamic programming. We consider dynamic programming in this section for the singly constrained case of (P) :

$$(P) \quad \min f(x) = \sum_{j=1}^n f_j(x_j)$$

$$\text{s.t. } g(x) = \sum_{j=1}^n g_j(x_j) \leq b$$

$$x \in X = X_1 \times X_2 \times \dots \times X_n,$$

where $X_j = \{x_j \in \mathbf{Z} | l_j \leq x_j \leq u_j\}$ with l_j and u_j being integers. We assume $g_j(x_j) \geq 0$ on X_j for all $j = 1, \dots, n$.

For adopting backward dynamic programming, the cost-to-go function is defined as follows,

$$\hat{t}_k(s) = \min \sum_{j=k}^n f_j(x_j)$$

$$\text{s.t. } s + \sum_{j=k}^n g_j(x_j) \leq b$$

$$x_j \in X_j, \quad j = k, \dots, n,$$

for $k = 1, \dots, n-1$, $s = 0, \dots, b$. The backward recursive equation is

$$\hat{t}_k(s) = \min \{f_k(x_k) + \hat{t}_{k+1}(s + g_k(x_k))\}$$

$$\text{s.t. } s + g_k(x_k) \leq b$$

$$x_k = l_k, \dots, u_k,$$

for $k = 1, \dots, n-1$, and $s = 0, \dots, b$, with boundary conditions

$$\hat{t}_k(s) = +\infty, \text{ for } s < 0, k = 1, \dots, n,$$

$$\hat{t}_n(s) = \min \{f_n(x_n) | s + g_n(x_n) \leq b, x_n = l_n, l_n + 1, \dots, u_n\}, s = 0, \dots, b.$$

For adopting forward dynamic programming, we define the following cost-to-accumulate function.

$$\begin{aligned} \tilde{t}_k(s) &= \min \sum_{j=1}^{k-1} f_j(x_j) \\ \text{s.t. } s + \sum_{j=1}^{k-1} g_j(x_j) &\leq s \\ x_j &\in X_j, \quad j = 1, \dots, k-1, \end{aligned}$$

The forward recursive equation is

$$\begin{aligned} \tilde{t}_k(s) &= \min \{f_k(x_k) + \tilde{t}_{k-1}(s - g_k(x_k))\} \\ \text{s.t. } g_k(x_k) &\leq s \\ x_k &= l_k, l_k + 1, \dots, u_k, \end{aligned}$$

for $k = 3, \dots, n$, $s = 0, \dots, b$.

In this situation, the dynamic programming table has a size of $n * (b + 1)$.

3.3. Surrogate dual search

A key issue in applying the surrogate dual method is how to solve the surrogate dual problem, more specifically, how to update the surrogate multipliers. Several surrogate dual search methods have been developed for linear integer programming and they can be also applied to nonlinear programming problems.

For $\alpha \in \mathbf{R}$, let $X(\alpha)$ denote the level set of $f(x)$, $X(\alpha) = \{x \in X | f(x) \leq \alpha\}$. For given $\mu \in \Lambda$ and $\alpha \in \mathbf{R}$, $v(P_\mu) \leq \alpha$ if and only if

$$S(\mu) \cap X(\alpha) \neq \emptyset, \quad (3.3.1)$$

where $S(\mu)$ is defined by (3.1.1). Consider the following problem

$$\begin{aligned} (P(\alpha, \mu)) \quad & \min \mu^T(g(x) - b) \\ \text{s.t. } & x \in X(\alpha). \end{aligned}$$

We notice that (3.3.1) holds if and only if $v(P(\alpha, \mu)) \leq 0$. Since $v(D_s^n) = \max\{v(P_\mu) | \mu \in \Lambda\}$, it follows that $v(D_s^n) \leq \alpha$ if and only if $v(P(\alpha, \mu)) \leq 0$ for all $\mu \in \Lambda$. Similar to the Lagrangian dual, we can define the following dual problem:

$$\begin{aligned} (D(\alpha)) \quad & \max v(P(\alpha, \mu)) \\ & \text{s.t. } \mu \in \Lambda. \end{aligned}$$

The above discussion leads to the following theorem.

Theorem 3.3.1. [30] For given $\alpha \in \mathbf{R}$, $v(D_s^n) \leq \alpha$ if and only if $v(D(\alpha)) \leq 0$.

An immediate corollary of Theorem 3.3.1 is as follows.

Corollary 3.3.1. [30] The optimal surrogate dual value $v(D_s^n)$ is the minimum $\alpha \in \mathbf{R}$ such that $v(D(\alpha)) \leq 0$.

The cutting plane method can be used to solve $D(\alpha)$. Notice that $D(\alpha)$ is equivalent to the following linear program:

$$\begin{aligned} & \max_{(\beta, \mu)} \beta \\ & \text{s.t. } \beta \leq \mu^T(g(x) - b), \forall x \in X(\alpha), \\ & \mu \in \Lambda. \end{aligned}$$

For each $x \in X(\alpha)$, the first constraint forms a cutting plane. We can construct $T^k \subset X(\alpha)$ step by step, thus approximating $v(D(\alpha))$ successively by solving the following linear program:

$$\begin{aligned} (LP_k) \quad & \max_{(\beta, \mu)} \beta \\ & \text{s.t. } \beta \leq \mu^T(g(x) - b), \forall x \in T^k, \\ & \mu \in \Lambda. \end{aligned}$$

Procedure 3.3.1. [30] (CUTTING PLANE PROCEDURE FOR (D_s^n))

Step 0 (Initialization). Set $\alpha^0 = -\infty$, $T^0 = \emptyset$. Choose any $\mu^1 \in \Omega$. Set $k = 1$.

Step 1 (Surrogate relaxation). Solve the surrogate relaxation problem (P_{μ^k}) and obtain an optimal solution x^k . If $g(x^k) \leq b$, stop and x^k is an optimal solution to (P) and $v(D_s^n) = v(P)$.

Step 2 (Updating lower bound). If $f(x^k) \geq \alpha^{k-1}$, then set $\alpha^k = f(x^k)$. Otherwise, set $\alpha^k = \alpha^{k-1}$.

Step 3 (Updating multiplier). Set $T^k = T^{k-1} \cup \{x^k\}$. Solve the linear program (LP_k) and obtain an optimal solution (β^k, μ^k) . If $\beta^k \leq 0$, stop and $\alpha^k = v(D_s^n)$. Otherwise, set $\mu^{k+1} = \mu^k$ and $k = k + 1$, go to Step 1.

Theorem 3.3.2. [30] Algorithm 3.3.1 finds an optimal value of D_s^n within a finite number of iterations.

To illustrate Procedure 3.3.1, we consider the following example:

Example 3.3.1.

$$\begin{aligned} \min f(x) &= 2x_1^2 - 3x_2^3 + 5x_3 \\ \text{s.t. } g_1(x) &= -2x_1 + 2x_2 + x_3^2 \leq 1, \\ g_2(x) &= x_1^3 + x_2 - x_3^2 \leq 0, \\ x_i &\in \{-1, 0, 1\}, i = 1, 2, 3. \end{aligned}$$

The iteration process of Procedure 3.3.1 for this example is described as follows:

Step 0. Set $\beta = 0, T^0 = \emptyset$. Choose $\mu^1 = (1, 0)^T$. Set $k = 1$.

Iteration 1

Step 1. Solve the surrogate problem

$$\begin{aligned} (P_{\mu^1}) \quad \min & 2x_1^2 - 3x_2^3 + 5x_3 \\ \text{s.t. } & 1 * (-2x_1 + 2x_2 + x_3^2) + 0 * (x_1^3 + x_2 - x_3^2) \leq 1, \\ & x_i \in \{-1, 0, 1\}, i = 1, 2, 3. \end{aligned}$$

We obtain $x^1 = (1, 1, -1)^T$ with $g(x^1) = (1, 1)^T$ not $\leq (1, 0)^T$.

Step 2. Since $f(x^1) = -6 > \alpha^0$, set $\alpha^1 = -6$.

Step 3. Set $T^1 = \{x^1\}$. Solve the linear program:

$$\begin{aligned}
 (LP_1) \quad & \max_{\beta, \mu} \beta \\
 \text{s.t.} \quad & \beta \leq 0 * \mu_1 + 1 * \mu_2, \\
 & \mu_1 + \mu_2 \leq 1, \\
 & \mu_1 \geq 0, \mu_2 \geq 0.
 \end{aligned}$$

We obtain $\beta^1 = 1 > 0$ and $\mu^1 = (0, 1)^T$. Set $k = 2$ and $\mu^2 = \mu^1$.

Iteration 2

Step 1. Solve the surrogate problem

$$\begin{aligned}
 (P_{\mu^1}) \quad & \min 2x_1^2 - 3x_2^3 + 5x_3 \\
 \text{s.t.} \quad & x_1^3 + x_2 - x_3^2 \leq 0, \\
 & x_i \in \{-1, 0, 1\}, i = 1, 2, 3.
 \end{aligned}$$

We obtain $x^2 = (-1, 1, -1)^T$ with $g(x^2) = (5, -1)^T \text{ not } \leq (1, 0)^T$.

Step 2. Since $f(x^2) = \alpha^1$, set $\alpha^2 = \alpha^1$.

Step 3. Set $T^2 = \{x^1, x^2\}$. Solve the linear program:

$$\begin{aligned}
 (LP_2) \quad & \max_{\beta, \mu} \beta \\
 \text{s.t.} \quad & \beta \leq \mu_2, \\
 & \beta \leq 4\mu_1 - \mu_2, \\
 & \mu_1 + \mu_2 \leq 1, \\
 & \mu_1 \geq 0, \mu_2 \geq 0.
 \end{aligned}$$

We obtain $\beta^2 = 2/3 > 0$ and $\mu^2 = (1/3, 2/3)^T$. Set $k = 3$ and $\mu^3 = \mu^2$.

Iteration 3

Step 1. Solve the surrogate problem

$$\begin{aligned}
 (P_{\mu^1}) \quad & \min 2x_1^2 - 3x_2^3 + 5x_3 \\
 \text{s.t.} \quad & 2x_1^3 - 2x_1 + 4x_2 - x_3^2 \leq 1, \\
 & x_i \in \{-1, 0, 1\}, i = 1, 2, 3.
 \end{aligned}$$

We obtain $x^3 = (0, 0, -1)^T$ with $g(x^2) = (1, -1)^T \leq (1, 0)^T$.

Stop, and we get the optimal solution to the problem. Note that the Lagrangian dual value is calculated as -6 which is smaller than -5 . It means that the surrogate dual method provides a dual value tighter than Lagrangian dual method does.

Compared to the original multiply constrained separable integer programming problem, the surrogate constraint formulation offers a promising platform with a singly constraint separable integer programming problem which dynamic programming can efficiently be applied. However, unless the optimal solution to the surrogate constraint formulation is feasible to the primal problem, there exists a duality gap due to the relaxation of the feasible region. On the other hand, performing surrogate dual search requires to apply dynamic programming many times, which will become a concern when the convergence is slow.

□ End of chapter.

CHAPTER 4

DISTANCE CONFINED PATH ALGORITHM

The shortest path problem deals with a task of finding the path with minimum time, distance, or cost from a source node to a destination node in a connected network. The shortest path problem has been playing a significant role in the development of operations research, due to its wide applications in various application areas, including transportation, communications networks, robot motion planning, and many others. The shortest path problem often serves as a starting point in learning dynamic programming as the philosophy of dynamic programming can be best explained by the shortest path problem. At the same time, many optimization problems solved by dynamic programming can be formulated under a unifying framework using a shortest path problem formulation, such as the knapsack problem and the sequence alignment problem in molecular biology [7, 9].

As a natural extension of the shortest path problem, the k th shortest paths problem [26, 42] is to find the shortest, the 2nd shortest, . . . , and the k th shortest paths connecting a given source-destination pair in a network. Development of algorithms for the k th shortest paths problem is motivated by considerations to incorporate additional constraints, model evaluation and generation of alternatives. Depending on whether cycles are allowed in the graph, there are two types of k th shortest paths network problems. Eppstein [10] provides a review on the

k th shortest looping path problem. The k th shortest loopless paths problem has been investigated in [42, 14, 22]. It is interesting to note from [42] that i) The j th shortest path can be obtained by comparing perturbations of the shortest, the 2nd shortest, \dots , and the $(j - 1)$ th shortest paths, and ii) the computational efforts of calculating k shortest paths is only linear with respect to k .

In this chapter, we investigate an algorithm in finding out paths of length within a given distance window for an n -stage loopless network with sink S and destination T . Let the network be denoted by (A, N) where N is the node set and A is the arc set. For a pair of two connecting nodes P and Q at neighboring stages, respectively, we use $d(P, Q)$ to denote the arc length between them. If there does not exist a direct arc between nodes P and Q , $d(P, Q)$ is set at infinity. Note that in our study, we allow multiple arcs with different lengths between a pair of two nodes, as we are not only interested in the shortest path. A path is an n -sector connecting path from S to T .

4.1. Yen's algorithm for the k th shortest path problem

There are several algorithms presently available for solving a k -shortest-loopless-paths problem in an M -node network. Yen [42] proposed an algorithm with complexity of $\frac{1}{2}kM^3$ which is linear with respect to k , which is one of the most efficient algorithms in finding the first k shortest paths in a loopless network. We focus in our study only on the n -stage network problems and we use the following notations and definitions modified from [42].

In an $(n + 1)$ -stage network, let $P^k = N_0^k \rightarrow N_1^k \rightarrow \dots \rightarrow N_n^k$ with $N_0^k = S$ and $N_n^k = T$ be the k th shortest path from S to T . For $i = 0, 1, \dots, n - 1$, let $D^{k-1}(i)$ be a path "deviated" from P^{k-1} , that satisfies: (i) Its first i nodes coincide with P^{k-1} ; and (ii) The distance from N_i^k to T is minimized subject to that the first $(i + 1)$ nodes of $D^{k-1}(i)$ do not coincide with the first $(i + 1)$ nodes

of any P^j , $j = 1, \dots, k - 1$. The subpath of $D^{k-1}(i)$ formed by the first i nodes is termed the root of $D^{k-1}(i)$ and is denoted by $R^{k-1}(i)$, while the subpath of $D^{k-1}(i)$ formed by the last $(n - i + 1)$ nodes is termed the spur of $D^{k-1}(i)$ and is denoted by $S^{k-1}(i)$.

The algorithm of Yen [42] for finding k shortest paths can be described now as follows:

Iteration 1: Determine P^1 , the shortest path from S to T , using dynamic programming or some other solution methods. Add P^1 to List A. Set $j = 2$.

Iteration j ($j = 2, 3, \dots, k$): Determine P^j . For $i = 1, 2, \dots, n - 1$, find possible $D^j(i)$, the paths deviated from P^{j-1} . If there are some, add them to List B. Choose the path in List B with the minimum distance and add it to List A as P^j . If $j = k$, stop; Otherwise, set $j = j + 1$ and go to Iteration j .

4.2. Application of Yen's method to integer programming

The surrogate dual formulation aggregates multiple constraints of problem (P) into a singly constrained formulation (P_μ). The singly constrained surrogate relaxation problem can be solved efficiently by dynamic programming. From the strong duality of the surrogate duality, one key recognition is that the optimal solution to (P) must be the solution in the ranking list of the minimum, the second minimum, ..., the k th minimum, ..., of (P_μ), that first satisfies the feasibility of (P).

Note that any singly constrained separable integer programming problem can be transformed to a shortest path problem in a loop-less graph. We can then apply Yen's method successively to find the k th shortest path of the corresponding loopless network which is first feasible in (P) among the first k shortest paths in the ranking list.

The network resulted from a surrogate constraint formulation is an n -stage

loopless network. The structure of the network is determined by the surrogate constraint, where the node at stage i corresponds to state which represents the accumulative “consumption” up to stage i with respect to the surrogate constraint. The range of the state at each stage can be determined by using (2.1.1) and (2.1.2).

The lengths of the arcs in the network are determined by the objective function. More specifically, the arc length between two nodes s_i and s_{i+1} is assessed by $\{f_i(x_i)|x_i \in X_i, g_i(x_i) = s_{i+1} - s_i\}$. If there are multiple x_i s satisfying $g_i(x_i) = s_{i+1} - s_i$ and $x_i \in X_i$, there are then multiple arcs between s_i and s_{i+1} with different arc lengths. Thus, any path in the network yields an objective value of $\{\sum_{i=1}^n f_i(x_i)|x_i \in X_i\}$.

We use the following example to illustrate the process of applying Yen’s method in solving integer programming problems.

Example 4.2.1.

$$\begin{aligned} \min f(x) &= 2x_1^2 - 3x_2^3 + 5x_3 \\ \text{s.t. } g_1(x) &= -2x_1 + 2x_2 - 2x_3^2 + x_3^2 \leq 0, \\ g_2(x) &= x_1^3 + x_2 - x_3^2 \leq 0, \\ g_3(x) &= x_1 + x_2 \leq 0, \\ x_i &\in \{-1, 0, 1\}, i = 1, 2, 3. \end{aligned}$$

Assigning $\mu = (0, 1, 0)'$ yields the following surrogate constraint formulation (P_μ) ,

$$\begin{aligned} \min f(x) &= 2x_1^2 - 3x_2^3 + 5x_3 \\ \text{s.t. } g_\mu(x) &= x_1^3 + x_2 - x_3^2 \leq 0, \\ x &\in X = \{-1, 0, 1\}^3. \end{aligned}$$

The problem can be transformed to a shortest path problem of a loop-less network in Figure 4.1. Let state be defined by

$$s_{i+1} = s_i + \sum_{j=1}^m \mu_j g_{ji}(x_i), \quad i = 0, 1, 2,$$

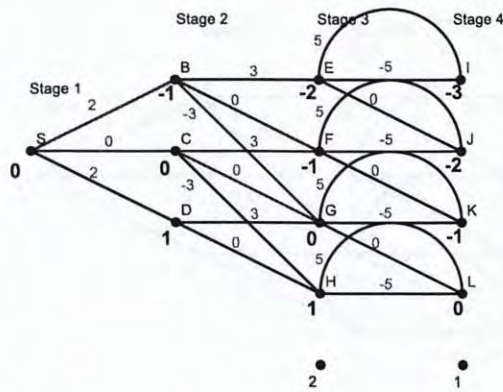


Figure 4.1: Network in Example 4.2.1

with $s_0 = 0$. According to (2.1.1) and (2.1.2), the state space is obtained: $GS_1 = [0, 0]$, $GS_2 = [-1, 1]$, $GS_3 = [-2, 1]$ and $GS_4 = [-3, 0]$. Each node in the network is marked with its corresponding state value. Note all paths which do not satisfy $g_\mu(x) \leq 0$ are removed from the graph.

We now show how to solve this integer programming example by using Yen's method iteration by iteration.

Iteration 0: Let $A = \emptyset$ and $B = \emptyset$.

Iteration 1: It is easy to verify that the shortest path is

$$P^1 = \{s = (0, 1, 0); x = ((0, 1, -1); f = -8\}.$$

The shortest path P^1 is described in Figure 4.2 with red lines.

Let $A = P^1 \cup A$. Since solution $(0, 1, -1)$ does not satisfy the first constraint of the primal problem, we go to the next iteration.

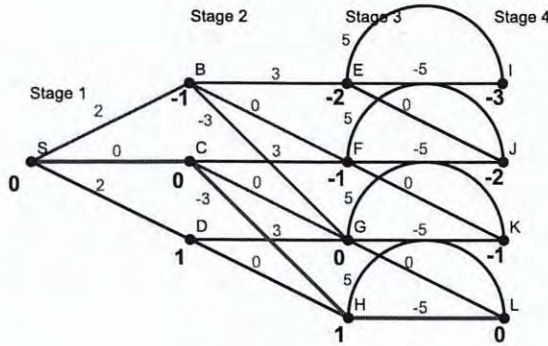


Figure 4.2: Shortest path in Example 4.2.1

Iteration 2: Considering the 3 deviations from P^1 gives rise to

$$D^1(2) = \{s = (0, 1, 0); x = (0, 1, 1); f = 2\},$$

$$D^1(1) = \{s = (0, 0, -1); x = (0, 0, -1), f = -5\},$$

$$D^1(0) = \{s = (-1, 0, -1); x = (-1, 1, -1); f = -6\}.$$

The 3 paths $D^1(0)$, $D^1(1)$, $D^1(2)$ deviating from the shortest path P^1 are marked in the Figure 4.3.

Let $P^2 = D^1(0)$, $A = P^2 \cup A$ and $B = \{D^1(2), D^1(1)\} \cup B$. As the 2nd shortest solution P^2 does not satisfy the 1st constraint of the primal problem, we go to the next iteration.

Iteration 3: The 3 deviations from P^2 give rise to

$$D^2(2) = \{s = (-1, 0, 0); x = (-1, 1, 0); f = -1\},$$

$$D^2(1) = \{s = (-1, -1, -2); x = (-1, 0, -1); f = -3\},$$

$$D^2(0) = \{s = (1, 1, -1); x = (1, 0, -1); f = -3\}.$$

Let $P^3 = D^1(1)$, $A = P^3 \cup A$ and $B = \{D^2(2), D^2(1), D^2(0)\} \cup (B \setminus D^1(1))$.

As the 3rd shortest solution P^3 does not satisfy the 1st constraint of the primal

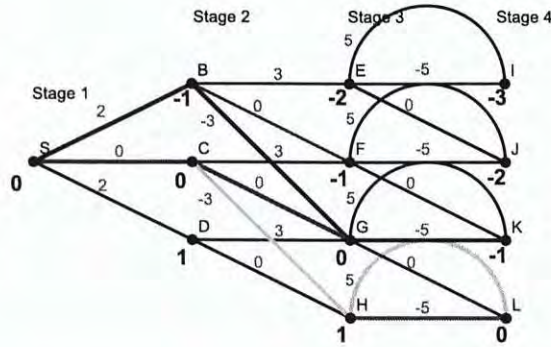


Figure 4.3: Deviating paths in Example 4.2.1

problem, we go to the next iteration.

Iteration 4: Considering the deviations from P^3 gives rise to

$$\begin{aligned}
 D^3(2) &= \{s = (0, 0, 0); x = (0, 0, 0); f = 0\}, \\
 D^3(1) &= \{s = (0, -1, -2); x = (0, -1, -1); f = -2\}, \\
 D^3(0) &= \{s = (1, 1, -1); x = (1, 0, -1); f = -3\}.
 \end{aligned}$$

Note that $D^3(0)$ is the same as $D^2(0)$, we do not need to add it to the candidate list. Let $P^4 = D^2(1)$, $A = P^4 \cup A$ and $B = \{D^3(2), D^3(1)\} \cup (B \setminus D^2(1))$. As the 4th shortest solution P^4 does not satisfy the 1st constraint of the primal problem, we go to the next iteration.

Iteration 5: Considering the deviations from P^4 gives rise to

$$\begin{aligned}
 D^4(2) &= \{s = (-1, -1, -1); x = (-1, 0, 0); f = 2\}, \\
 D^4(1) &= \{s = (-1, -2, -3); x = (-1, -1, -1); f = 0\}, \\
 D^4(0) &= \{s = (1, 1, -1); x = (1, 0, -1); f = -3\}.
 \end{aligned}$$

Note that $D^4(0)$ is the same as $D^3(0)$, we do not need to add it to the candidate list. Let $P^5 = D^2(0)$, $A = P^5 \cup A$ and $B = \{D^4(2), D^4(1)\} \cup (B \setminus D^2(0))$. As

the 5th shortest solution P^5 does not satisfy the 1st constraint of the primal problem, we go to the next iteration.

Iteration 6: Considering the deviations from P^5 gives rise to

$$\begin{aligned} D^5(2) &= \{s = (1, 1, 0); x = (1, 0, 1); f = 7\}, \\ D^5(1) &= \{s = (1, 0, -1); x = (1, -1, -1); f = 0\}. \end{aligned}$$

Note that there is no candidate for $D^5(0)$. Let $P^6 = D^3(1)$, $A = P^6 \cup A$ and $B = \{D^5(2), D^5(1)\} \cup (B \setminus D^3(1))$. As the 6th shortest solution P^6 does not satisfy the 1st constraint of the primal problem, we go to the next iteration.

Iteration 7: Considering the deviations from P^6 gives rise to

$$D^6(2) = \{s = (0, -1, -1); x = (0, -1, 0); f = 3\}.$$

Note that there is no candidate for either $D^6(1)$ or $D^6(0)$. Let $P^7 = D^2(2)$, $A = P^7 \cup A$ and $B = \{D^6(2)\} \cup (B \setminus D^2(2))$. As the 7th shortest solution P^7 does not satisfy the 1st constraint of the primal problem, we go to the next iteration.

Iteration 8: Considering the deviations from P^7 gives rise to

$$\begin{aligned} D^7(2) &= \{s = (-1, 0, -1); x = (-1, 1, 1); f = 4\}, \\ D^7(1) &= \{s = (-1, -2, -3); x(-1, -1, -1); f = 0\}. \end{aligned}$$

As there is no candidate for $D^7(0)$ and $D^7(1)$ is the same as $D^3(0)$, we do not have new member to add to the candidate list. Let $P^8 = D^4(1)$, $A = P^8 \cup A$ and $B = \{D^7(2), D^7(1)\} \cup (B \setminus D^3(2))$. As the 8th shortest solution P^8 satisfies all the constraints of the primal problem, $x = (0, 0, 0)$ is the optimal solution to Example 4.2.1. To reach the 8th shortest path, we have generated in total 16 paths in the process.

All the paths can be found according to Yen's algorithm from the corresponding dynamic programming table in Table 4.1 where $\hat{t}(s_i, x_i), i = 1, \dots, 3$, denotes the minimum distance from state s_i to destination using control x_i . For example, starting from $s_3 = -2$, three shortest subpaths from state $s_3 = -2$

to destination using controls $x_3 = -1, 0$ and 1 arise respectively. Similarly, we record the values for $s_3 = -1, 0$ and 1 . Then we calculate the minimum values using different controls for every state s_2 and finally obtain 3 shortest paths from $s_1 = 0$ to destination using $x_1 = -1, 0$, and 1 . The shortest path $P^1 = \{s = (0, 1, 0); x = ((0, 1, -1); f = -8\}$ is marked with “*”. Then we can find the candidates for the second shortest path according to Yen’s algorithm. To find out $D^1(0)$, the distance from $s_1 = 0$ to destination should be minimized and the first control of $D^1(0)$ should be different from that of P^1 ’s. Taking control $x_1 = -1$ and moving forward we get $D^1(0) = \{s = (-1, 0, -1); x = (-1, 1, -1); f = -6\}$. For $D^1(1)$, the first node is required to be the same as that of P^1 ’s, which means that $x_1 = 0$. Then we need to find the minimum distance from $s_2 = 0$ to destination with a control different from that of P^1 ’s. Deviating from $x_2 = 1$, we find $x_2 = 0$ can satisfy our requirements. Moving forward we obtain $D^1(1) = \{s = (0, 0, -1); x = (0, 0, -1), f = -5\}$. At last, to find out $D^1(2)$, the first two nodes should be the same as that of P^1 ’s, using $x_1 = 0$ and $x_2 = 1$ leads to $s_3 = 1$. Then the distance from $s_3 = 1$ to destination should be minimized using control different from $x_3 = -1$. We finally get $D^1(2) = \{s = (0, 1, 0); x = (0, 1, 1); f = 2\}$.

4.3. Distance confined path problem

Although Yen’s algorithm for finding the k th shortest path possesses a linear complexity with respect k , it acquires identification from the shortest to the $(k - 1)$ th shortest before reaching the k th shortest. When k is large, it is often unnecessary to go through this tedious process. In real life, with an appointment being \bar{t} hours away, a traveler may want to find a path which consumes touring time within a time window $[\bar{t} - 2\delta, \bar{t} - \delta]$, where δ is a positive small number, to fully utilize the available time for sightseeing. In the later part of this chapter, we will see that a surrogate constraint formulation of multiply constrained integer programming problem corresponds to a network problem where the desired path

Table 4.1: The extended dynamic programming table in Example 4.2.1

k	1				2				3			
S_k	x_1	$f_1(x_1)$	s_2	$\hat{t}(s_1, x_1)$	x_2	$f_2(x_2)$	s_3	$\hat{t}(s_2, x_2)$	x_3	$f_3(x_3)$	s_4	$\hat{t}(s_3, x_3)$
1					1	-3	2	∞	1	5	0	5
					0	0	1	-5	0	0	1	∞
					-1	3	0	-2	-1*	-5	0	-5*
0	1	2	1	-3	1*	-3	1	-8*	1	5	-1	5
	0*	0	0	-8*	0	0	0	-5	0	0	0	0
	-1	2	-1	-6	-1	3	-1	-2	-1	-5	-1	-5
-1					1	-3	0	-8	1	5	-2	5
					0	0	-1	-5	0	0	-1	0
					-1	3	-2	-2	-1	-5	-2	-5
-2									1	5	-3	5
									0	0	-2	0
									-1	-5	-3	-5

is bounded from below by a distance value dictated by the dual value of the integer programming problem. Thus, we are interested in developing solution algorithms for a distance confined path problem introduced below.

Definition 4.3.1. *The distance confined path problem is to find all the paths in a graph from the origin, S , to the destination, T , with distance within a given distance window $[l, u]$.*

We use $D_{\min}(P, Q)$ and $D_{\max}(P, Q)$ to denote the minimum distance and the maximum distance, respectively, between two nodes P and Q in the graph. More specifically, $D_{\min}(S, P)$ denotes the minimum distance from the origin to node P , $D_{\max}(S, P)$ the maximum distance from the origin to node P , $D_{\min}(P, T)$ the minimum distance from node P to the destination, and $D_{\max}(S, P)$ the maximum distance from node P to the destination.

Definition 4.3.2. *The distance confined path problem is infeasible either when $D_{\max}(S, T) \leq l$ or when $D_{\min}(S, T) \geq u$.*

Carrying out i) forward dynamic programming twice to find out the shortest path and the longest path from the origin to the destination, and ii) backward dynamic programming twice to find out the shortest path and the longest path from the origin to the destination yields quadruple $(D_{\min}(S, P), D_{\max}(S, P), D_{\min}(P, T), D_{\max}(P, T))$ for every node P in the graph.

If $D_{\min}(S, P) + D_{\min}(P, T) > u$, or $D_{\max}(S, P) + D_{\max}(P, T) < l$, then any path passing through node P possesses a path length outside of the range $[l, u]$ and node P can be removed from the graph from further consideration.

Let us consider the feasible range of the accumulative path length from the origin to node P in a distance confined path problem. We define the following pair for every node P in the graph,

$$\underline{L}(P) = \max\{D_{\min}(S, P), l - D_{\max}(P, T)\} \quad (4.3.1)$$

and

$$\bar{L}(P) = \min\{D_{\max}(S, P), u - D_{\min}(P, T)\}. \quad (4.3.2)$$

When a distance confined path problem is feasible, we have $\bar{L}(S) = \underline{L}(S) = 0$ for the origin. When both $D_{\min}(S, T) \leq l$ and $D_{\max}(S, T) \geq u$, we have $\bar{L}(T) = u$ and $\underline{L}(T) = l$ for the destination.

Lemma 4.3.1. *For any node P in the graph, $\bar{L}(P) < \underline{L}(P)$ holds if and only if $D_{\max}(S, P) + D_{\max}(P, T) < l$ or $D_{\min}(S, P) + D_{\min}(P, T) > u$.*

Proof: i) Assume that $\bar{L}(P) < \underline{L}(P)$.

(a) If $D_{\min}(S, P) \geq l - D_{\max}(P, T)$, then $\underline{L}(P) = D_{\min}(S, P) > \bar{L}(P) = \min\{D_{\max}(S, P), u - D_{\min}(P, T)\}$, which can only happen when $D_{\min}(S, P) > u - D_{\min}(P, T)$.

(b) If $D_{\min}(S, P) \leq l - D_{\max}(P, T)$, then $\underline{L}(P) = l - D_{\max}(P, T) > \bar{L}(P) = \min\{D_{\max}(S, P), u - D_{\min}(P, T)\}$, which can only happen when $l - D_{\max}(P, T) > D_{\max}(S, P)$.

ii) Assume that $D_{\max}(S, P) + D_{\max}(P, T) < l$. Then $D_{\min}(S, P) \leq D_{\max}(S, P) < l - D_{\max}(P, T)$. Thus, $\underline{L}(P) = l - D_{\max}(P, T)$. As $D_{\max}(S, P) < l - D_{\max}(P, T) \leq u - D_{\min}(P, T)$, then $\bar{L}(P) = D_{\max}(S, P)$, leading to $\bar{L}(P) < \underline{L}(P)$.

iii) Assume that $D_{\min}(S, P) + D_{\min}(P, T) > u$. Then $D_{\max}(S, P) \geq D_{\min}(S, P) > u - D_{\min}(P, T)$. Thus, $\bar{L}(P) = u - D_{\min}(P, T)$. As $D_{\min}(S, P) > u - D_{\min}(P, T) \geq l - D_{\max}(P, T)$, then $\underline{L}(P) = D_{\min}(S, P)$, leading to $\bar{L}(P) < \underline{L}(P)$. \square

Proposition 4.3.1. *Graph reduction rule*

i) Any node P with $\bar{L}(P) < \underline{L}(P)$ can be removed from the graph.

ii) Any branch between two connected nodes P and Q with $D_{\max}(S, P) + d(P, Q) < \underline{L}(Q)$ or $D_{\min}(S, P) + d(P, Q) > \bar{L}(Q)$ can be removed from the graph.

Proof: i) Evidenced from Lemma 4.3.1, any path passing through P with $\bar{L}(P) < \underline{L}(P)$ has a length outside of range $[l, u]$ and node P can be thus removed.

ii) Assume that $D_{\max}(S, P) + d(P, Q) < \underline{L}(Q)$. Since $D_{\max}(S, P) + d(P, Q) \geq D_{\min}(S, P) + d(P, Q) \geq D_{\min}(S, Q)$, then $\underline{L}(Q) = l - D_{\max}(Q, T)$. Thus, $D_{\max}(S, P) + d(P, Q) < l - D_{\max}(Q, T)$, i.e., $D_{\max}(S, P) + d(P, Q) + D_{\max}(Q, T) < l$. As the length of the longest path passing through branch (P, Q) is still smaller than l , branch (P, Q) can be removed from the graph.

(ii) Assume that $D_{\min}(S, P) + d(P, Q) > \bar{L}(Q)$. Since $D_{\min}(S, P) + d(P, Q) \leq D_{\max}(S, P) + d(P, Q) \leq D_{\max}(S, Q)$, then $\bar{L}(Q) = u - D_{\min}(Q, T)$. Thus, $D_{\min}(S, P) + d(P, Q) > u - D_{\min}(Q, T)$, i.e., $D_{\min}(S, P) + d(P, Q) + D_{\min}(Q, T) > u$. As the length of the shortest path passing branch (P, Q) is still larger than u , branch (P, Q) can be removed from the graph. \square

Definition 4.3.3. *If all incoming or all outgoing branches of a node P (other than S or T) are removed, node P is called isolated.*

Any isolated node can be removed from the graph. For any node P in the graph, after we identify and record all the paths which pass through P and have a length within range $[l, u]$, node P can be removed from the graph.

Algorithm 4.3.1. Finding all paths with length in a given range $[l, u]$.

Step 1. For every node p in the graph, calculate $D_{\min}(S, P)$, $D_{\max}(S, P)$, $D_{\min}(P, T)$, $D_{\max}(P, T)$.

Step 2. Update l and u by

$$l = \max\{l, D_{\min}(S, T)\}$$

$$u = \min\{u, D_{\max}(S, T)\}$$

Calculate $\underline{L}(P)$ and $\bar{L}(P)$ for all nodes. Remove infeasible nodes and branches according to Proposition 4.3.1 and remove isolated nodes.

Step 3. Update the graph and repeat the process in Steps 1 and 2 until no further reduction can be done. If there is no node left, stop. Otherwise, go to Step 4.

Step 4. Finding node \tilde{P} such that $[(D_{\min}(S, \tilde{P}) + D_{\min}(\tilde{P}, T)), (D_{\max}(S, \tilde{P}) + D_{\max}(\tilde{P}, T))]$ has the minimum intersection with $[l, u]$ among all $[(D_{\min}(S, P) + D_{\min}(P, T)), (D_{\max}(S, P) + D_{\max}(P, T))]$. Use Yen's k th shortest or longest path algorithm to find out all paths passing through node \tilde{P} that have length range within $[l, u]$ and record them. Remove node \tilde{P} from the graph. Go back to Step 1.

It is obvious from the above algorithm that, for all nodes in the graph, $\underline{L}(P)$ and $\bar{L}(P)$ are nondecreasing and nonincreasing, respectively, in the iterations when implementing Algorithm 4.3.1.

Example 4.3.1. Find all the paths with path length within range $[28, 30]$ in Figure 4.4.

Performing dynamic programming four times gives rise to Table 4.2 in which the quadruples are listed for every node in the graph. From Table 4.2, node F can be removed as $D_{\max}(S, F) + D_{\max}(F, T) < 28$ or $\underline{L}(F) > \bar{L}(F)$. Furthermore, we find that $D_{\max}(S, B) + d(B, E) < \underline{L}(E)$, $D_{\max}(S, C) + d(C, E) < \underline{L}(E)$, $D_{\min}(S, D) + d(D, E) > \bar{L}(E)$. Thus, all incoming branches to node E can be removed. Node E becomes isolated and can be removed too. The problem structure of Example

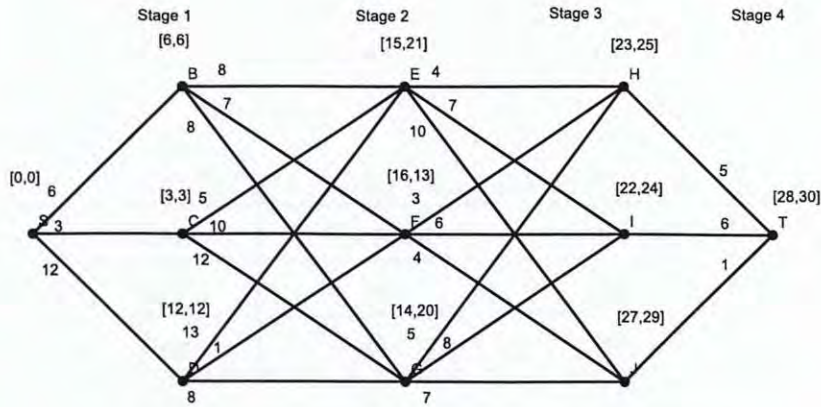


Figure 4.4: Graph in Example 4.3.1

4.3.1 reduces to a one in Figure 4.5 and the value table is updated as in Table 4.3.

As $[(D_{\min}(S, B) + D_{\min}(B, T)), (D_{\max}(S, B) + D_{\max}(B, T))] = [22, 28]$ reaches the minimum intersection with $[l, u] = [28, 30]$ among all nodes in Figure 4.5, we start from node B . Using Yen's algorithm, we first find the longest path passing node B with length 28, SBGIT. As the second longest path passing node B , SBGHT, has a length 24, we conclude all paths passing node B , except SBGIT, have lengths less than 28. After recording SBGIT, we remove node B . Similar situation happens at node D . The shortest path passing node D , SDGJT, has path length 28 and the 2nd shortest path passing node D , SDGHT, has path length 30, and the distance of the 3rd shortest path passing node D goes beyond 30. After recording paths SDGJT and SDGHT, node D can also be removed and paths are recorded. After removing node B and D and checking the remaining 4 paths, only one path has a distance length within $[28, 30]$, i.e., path SCGIT with length 29. In summary, our algorithm successively identifies four paths with length in $[28, 30]$.

Table 4.2: The value table in Example 4.3.1

Node	Value	Forward	Backward	\underline{sp}	\overline{sp}	Total
B	min	6	12	6	6	18
	max	6	22			28
C	min	3	14	3	3	17
	max	3	26			29
D	min	12	6	12	12	18
	max	12	26			38
E	min	8	9	15	21	17
	max	25	13			38
F	min	13	5	16	13	18
	max	13	12			25
G	min	14	8	14	20	22
	max	20	14			34
H	min	12	5	23	25	17
	max	29	5			34
I	min	15	6	22	24	21
	max	32	6			38
J	min	17	1	27	29	18
	max	35	1			36

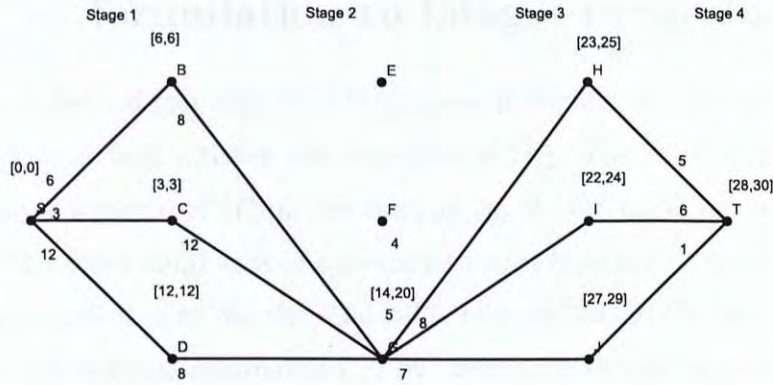


Figure 4.5: Graph in Example 4.3.1

Table 4.3: The updated value table in Example 4.3.1

Node	Value	Forward	Backward	\underline{sp}	\overline{sp}	Total
B	min	6	16	6	6	22
	max	6	22			28
C	min	3	20	3	3	23
	max	3	26			29
D	min	12	16	12	12	28
	max	12	22			34
G	min	14	8	14	20	22
	max	20	14			34
H	min	19	5	23	25	24
	max	25	5			30
I	min	22	6	22	24	28
	max	28	6			34
J	min	21	1	27	29	22
	max	27	1			28

4.4. Application of distance confined path formulation to integer programming

Yen's method can serve for the purpose in finding the k th minimum solution of (P_μ) , that first satisfies the feasibility of (P) . The value of k to reach the first feasible solution of (P) in the ranking list of (P_μ) could be, however, very large. On the other hand, it is unnecessary to start from the minimum solution of (P_μ) . One shortcut is to use the dual value to lower-bound the optimal value. We can find the optimal solution of (P) by identifying the ranking list of solutions only with objective values within range $[\underline{f}, \bar{f}]$ where \underline{f} is the dual value of problem (P) and \bar{f} is the objective value of the incumbent. Furthermore, we can partition range $[\underline{f}, \bar{f}]$ into a union of several non-overlapping sub-ranges to speed-up the convergence. When a feasible solution of (P) is found in a sub-range with lower objective values, there will be no need to search in any sub-range with high objective values.

For the graph defined by $g_\mu(x) \leq b_\mu$, we can further utilize the constraint $g_j(x) \leq b_j$, $j = 1, \dots, m$, to remove infeasible nodes and branches. More specifically, we consider the following m feasibility problems for $j = 1, \dots, m$,

$$(F_{\mu j}) \quad g_\mu(x) \leq b_\mu \text{ and } g_j(x) \leq b_j.$$

Problem $(F_{\mu j})$ can be partially solved by the distance confined path problem formulation which we discussed in the previous section. We first construct a graph based on $g_\mu(x) \leq b_\mu$ and assign arc length according to $g_j(x)$. Utilizing the constraint of $g_j(x) \leq b_j$, we may remove some infeasible nodes and branches in the graph. Problem $(F_{\mu j})$, $j = 1, \dots, m$, can be solved successively, one after the other. The reduced graph from the previous formulation is used as the starting point for the next formulation.

We consider the integer programming problem of example 4.2.1 again:

$$\begin{aligned} \min f(x) &= 2x_1^2 - 3x_2^3 + 5x_3 \\ \text{s.t. } g_1(x) &= -2x_1 + 2x_2 - 2x_2^3 + x_3^2 \leq 0, \\ g_2(x) &= x_1^3 + x_2 - x_3^2 \leq 0, \\ g_3(x) &= x_1 + x_2 \leq 0, \\ x_i &\in \{-1, 0, 1\}, i = 1, 2, 3. \end{aligned}$$

It can be verified that the optimal solution to this example is $(0, 0, 0)$. On the other hand, the surrogate dual search yields a dual value of -1 . We demonstrate now how our proposed solution scheme works.

Setting $\mu = (0, 1, 0)'$, we formulate a surrogate constraint problem (P_μ) as follows,

$$\begin{aligned} \min f(x) &= 2x_1^2 - 3x_2^3 + 5x_3 \\ \text{s.t. } g_2(x) &= x_1^3 + x_2 - x_3^2 \leq 0, \\ x &\in X = \{-1, 0, 1\}^3. \end{aligned}$$

Using Yen's method, we can identify the ranking list successively starting from the minimum solution, in which the first feasible solution, $(0, 0, 0)$, is reached at the 8th position. Our proposed method will, instead, identify the ranking list of $f(x)$ within the range $[-1, 10]$, where -1 is the dual value and 10 is an upper bound obtained by assigning the largest value to each term in $f(x)$. We further partition the whole objective value range into 4 sub-ranges $[-1, 1]$, $[2, 4]$, $[5, 7]$ and $[8, 10]$ and to check the solutions in the lowest range $[-1, 1]$ first.

But before we solve the above surrogate constraint problem to its optimum, we may first use other constraints of the primal formulation to reduce the graph dictated by $g_2(x) \leq 0$. Problem $(F_{\mu_1}) : \{g_2(x) \leq 0 \text{ and } g_1(x) \leq 0\}$ is depicted in Figure 4.6 in which the graph structure is constructed based on $g_2(x) \leq 0$ and the arc length is assigned according to $g_1(x)$.

The corresponding value table is given in Table 4.4. Take $g_1(x) \leq 0$ into consideration, we can remove nodes and arcs which only allow paths with positive

Table 4.4: The original value table of $F_{\mu 1}$ in Example 4.2.1

Node	Value	Forward	Backward	sp	\overline{sp}	Total
B	min	2	0	2	0	2
	max	2	1			3
C	min	0	0	0	0	0
	max	0	1			1
D	min	-2	0	-2	-2	-2
	max	-2	1			-1
E	min	2	0	2	0	2
	max	2	1			3
F	min	0	0	0	0	0
	max	2	1			3
G	min	-2	0	-2	0	-2
	max	0	1			1
H	min	-2	1	-2	0	-1
	max	0	1			1
I	min	3	0	3	0	3
	max	3	0			3
J	min	1	0	1	0	1
	max	2	0			2
K	min	-1	0	-1	0	-1
	max	2	0			2
L	min	-2	0	-2	0	-2
	max	2	0			2

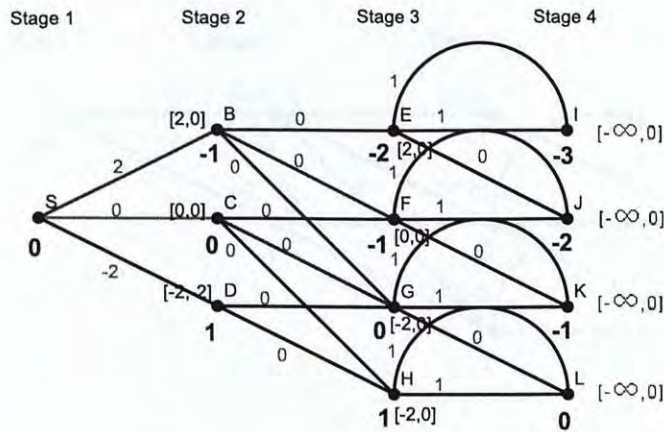


Figure 4.6: Graphical presentation of F_{μ_1}

g_1 values. Nodes B , E , I , and J can be removed as their $\underline{L}(P) > \bar{L}(P)$. The graph is reduced to Figure 4.7 and the corresponding revised value table is in given in Table 4.5.

As the graph in Figure 4.7 can not be further reduced by constraint $g_1(x) \leq 0$ and solving problem $(F_{\mu_2}) : \{g_2(x) \leq 0 \text{ and } g_2(x) \leq 0\}$ would not help removing infeasible nodes or branches in the graph constructed by $g_2(x) \leq 0$, we now switch to $(F_{\mu_3}) : \{g_2(x) \leq 0 \text{ and } g_3(x) \leq 0\}$ on the reduced graph, resulting in the graph in Figure 4.8. Reading corresponding value in Table 4.6 indicates that node H can be removed, leading to the graph in Figure 4.9 and the corresponding value table in Table 4.7.

The graph in Figure 4.9 cannot be further reduced by constraint $g_3(x) \leq 0$. We now switch to the surrogate constraint formulation with $f(x) \in [-1, 1]$, resulting in the graph in Figure 4.10 and the corresponding value table in Table 4.8.

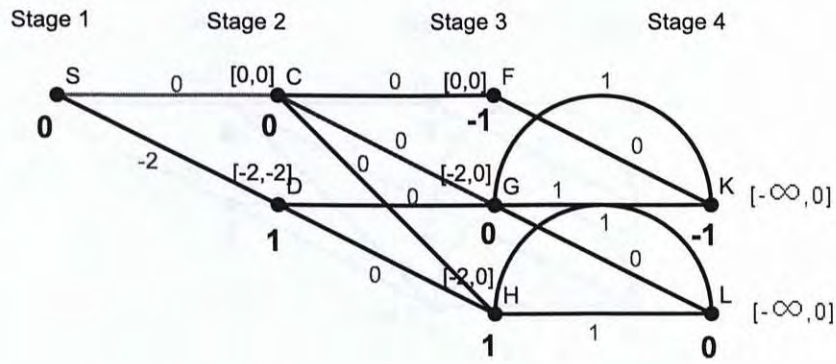


Figure 4.7: Graphical presentation of reduced version 1 of $F_{\mu 1}$

Table 4.5: The updated value table of $F_{\mu 1}$ in Example 4.2.1

Node	Value	Forward	Backward	\underline{sp}	\overline{sp}	Total
C	min	0	0	0	0	0
	max	0	1			1
D	min	-2	0	-2	-2	-2
	max	-2	1			-1
F	min	0	0	0	0	0
	max	0	0			0
G	min	-2	2	-2	0	-2
	max	0	1			1
H	min	-2	1	-2	0	-1
	max	0	1			1
K	min	-1	0	-1	0	-1
	max	1	0			1
L	min	-2	0	-2	0	-2
	max	2	0			2

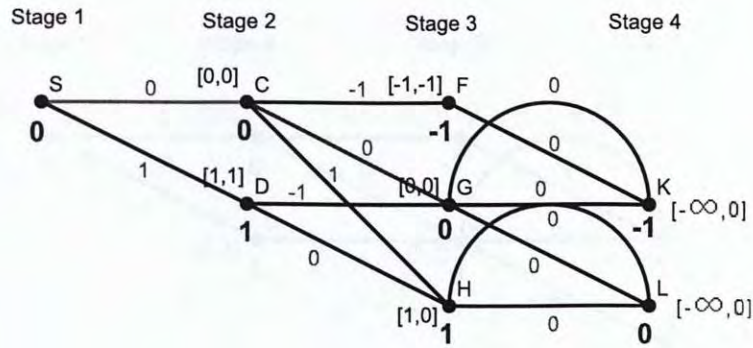


Figure 4.8: Graphical presentation of reduced version 2 of $F_{\mu 3}$

Table 4.6: The original value table of $F_{\mu 3}$ in Example 4.2.1

Node	Value	Forward	Backward	\underline{sp}	\overline{sp}	Total
C	min	0	-1	0	0	-1
	max	0	1			1
D	min	1	-1	1	1	0
	max	1	0			1
F	min	-1	0	-1	-1	-1
	max	-1	0			-1
G	min	0	0	0	0	0
	max	0	0			0
H	min	1	0	1	0	1
	max	1	0			1
K	min	-1	0	-1	0	-1
	max	0	0			0
L	min	0	0	0	0	0
	max	1	0			1

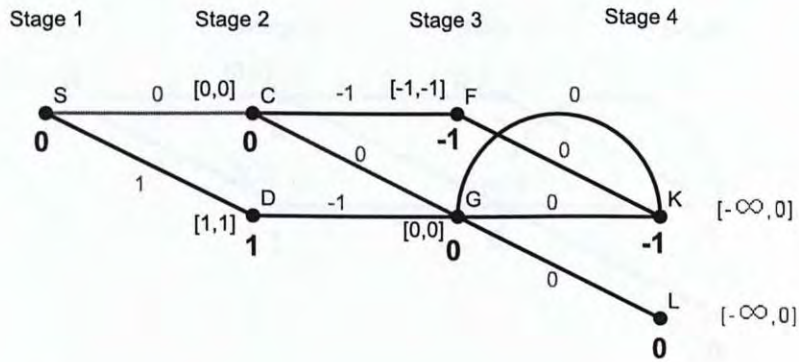


Figure 4.9: Graphical presentation of reduced version 3 of $F_{\mu 3}$

Table 4.7: The updated value table of $F_{\mu 3}$ in Example 4.2.1

Node	Value	Forward	Backward	\underline{sp}	\overline{sp}	Total
C	min	0	-1	0	0	-1
	max	0	0			0
D	min	1	-1	1	1	0
	max	1	-1			0
F	min	-1	0	-1	-1	-1
	max	-1	0			-1
G	min	0	0	0	0	0
	max	0	0			0
K	min	-1	0	-1	0	-1
	max	0	0			0
L	min	0	0	0	0	0
	max	0	0			0

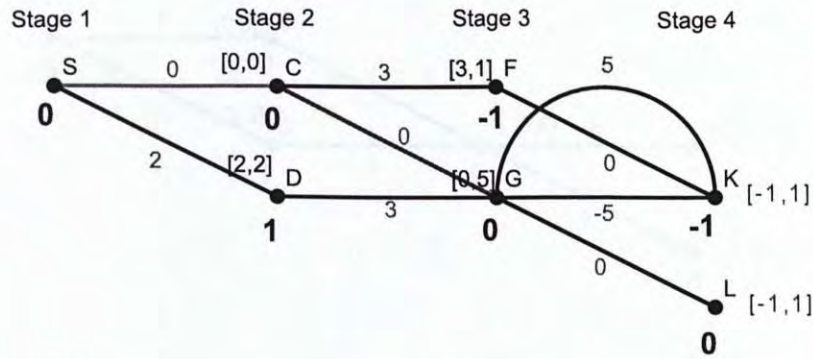


Figure 4.10: Graphical presentation of reduced version 4 of P_μ

Table 4.8: The updated value table of P_μ in Example 4.2.1

Node	Value	Forward	Backward	\underline{sp}	\overline{sp}	Total
C	min	0	-5	0	0	-5
	max	0	5			5
D	min	2	-2	2	2	0
	max	2	8			10
F	min	3	0	3	1	3
	max	3	0			3
G	min	0	-5	0	5	-5
	max	5	5			10
K	min	-5	0	-1	1	-5
	max	10	0			10
L	min	0	0	0	1	0
	max	5	0			5

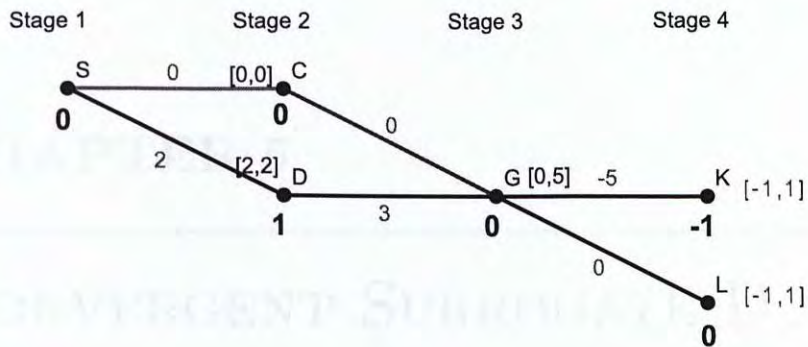


Figure 4.11: Graphical presentation of reduced version 5 of P_μ

After checking the value table in Table 4.8, we conclude that i) node F can be removed as $\underline{L}(F) > \bar{L}(F)$ and ii) one branch between nodes G and K with arc length of 5 can be removed as $D_{\min}(S, G) + 5 > \bar{L}(K)$. The graph is further reduced to Figure 4.11. Checking the feasibility of the four possible paths identifies the optimal solution $(0, 0, 0)$.

□ End of chapter.

CHAPTER 5

CONVERGENT SURROGATE DUAL SEARCH

The most challenging task to achieve the strong duality in the surrogate constraint formulation is to modify the formulation of (P_μ) such that the optimal solution of the modified (P_μ) is feasible in the primal (P) at the same time.

The feasible region of (P_μ) , $S(\mu)$, enlarges the feasible region of (P) , S . When an infeasible solution of (P) that has an objective value smaller than $v(P)$ is included in $S(\mu)$, the optimal solution of (P_μ) cannot be feasible. The solution concept presented in this chapter is to remove such infeasible points which attain the optimality of (P_μ) from further consideration. We require the integrality of f in this research task in order to efficiently implement dynamic programming.

Consider the following modified version of (P) by imposing a lower cut α and an upper cut β :

$$\begin{aligned} (P(\alpha, \beta)) \quad & \min f(x) \\ & \text{s.t. } g_i(x) \leq b_i, \quad i = 1, \dots, m, \\ & x \in X(l, u) = \{x \in X \mid \alpha \leq f(x) \leq \beta\}. \end{aligned}$$

It is obvious that $(P(\alpha, \beta))$ is equivalent to (P) if $\alpha \leq f^* \leq \beta$.

The surrogate relaxation of $(P(\alpha, \beta))$ is:

$$\begin{aligned}
 (P_\mu(\alpha, \beta)) \quad & \min f(x) \\
 \text{s.t.} \quad & \mu^T(g(x) - b) \leq 0 \\
 & \alpha \leq f(x) \leq \beta \\
 & x \in X.
 \end{aligned}$$

Problem $(P_\mu(\alpha, \beta))$ is a separable integer programming problem with two constraints, one of which is the same as the objective function. Utilizing this special property, we will develop an efficient solution scheme in solving $(P_\mu(\alpha, \beta))$.

Note that problem $(P_\mu(\alpha, \beta))$ is equivalent to the conventional surrogate constraint formulation (P_μ) when $\alpha \leq v(P_\mu)$. Let the feasible region of $(P_\mu(\alpha, \beta))$ be $S(\mu; (\alpha, \beta))$. The following theorem provides the basis for development of the convergent surrogate dual search using the concept of an objective cut.

Theorem 5.0.1. (i) When $v(P_\mu) \leq \alpha \leq v(P)$, $S \subseteq S(\mu; (\alpha, \beta)) \subseteq S(\mu)$.

(ii) Let $\delta = \min\{v(P) - f(x) \mid x \in X \text{ and } f(x) < v(P)\}$. Any optimal solution to problem (P) also solves problem $(P_\mu(\alpha, \beta))$ when $v(P) - \delta < \alpha \leq v(P)$.

The extent of the initial interval $[\alpha, \beta]$ has significant impact on the efficiency of dynamic programming when solving $P_\mu(\alpha, \beta)$. To reduce the range without losing any optimal solution, a partition scheme is proposed to divide the range $[\alpha, \beta]$ into q smaller non-overlapping blocks such that

$$[\alpha, \beta] = \cup_{s=1}^q [\alpha^s, \beta^s],$$

where $\alpha^1 = \alpha$, $\beta^q = \beta$ and $\alpha^{s+1} = \beta^s + 1$. The original problem can be then divided into q subproblems with $s = 1, 2, \dots, q$:

$$\begin{aligned}
 (P^s) \quad & \min f(x) \\
 \text{s.t.} \quad & g_i(x) \leq b_i, \quad i = 1, \dots, m, \\
 & \alpha^s \leq f(x) \leq \beta^s.
 \end{aligned} \tag{5.0.1}$$

These q subproblems will be solved successively from $s = 1$ to $s = q$. If an optimal solution x^* is found in problem (P^s) for $1 \leq s \leq q$, then x^* is an optimal solution to (P) and no need to solve the remaining subproblems. If all the q subproblems are infeasible, then we claim the infeasibility of the primal problem.

Let \bar{s} and \underline{s} denote the upper bound and lower bound of the range of state variable s_j , respectively. Let

$$\begin{aligned}\bar{f}_j &= \max_{l_j \leq x_j \leq u_j} f_j(x_j), \\ \underline{f}_j &= \min_{l_j \leq x_j \leq u_j} f_j(x_j).\end{aligned}$$

With the initial condition $\bar{s}_1^F = \underline{s}_1^F = 0$, the range s_j^F of the state variable s_j at stage j can be determined by a forward recursive formulation,

$$\begin{aligned}\bar{s}_{j+1}^F &= \bar{s}_j^F + \bar{f}_j, \quad \text{for } j = 1, \dots, n, \\ \underline{s}_{j+1}^F &= \underline{s}_j^F + \underline{f}_j, \quad \text{for } j = 1, \dots, n.\end{aligned}$$

With the initial condition $\bar{s}_1^B = u_k$, $\underline{s}_1^B = l_k$, the range s_j^B of the state variable s_j at stage j can be determined by a backward recursive formulation,

$$\begin{aligned}\bar{s}_j^B &= \bar{s}_{j+1}^B - \underline{f}_j, \quad \text{for } j = n, \dots, 1, \\ \underline{s}_j^B &= \underline{s}_{j+1}^B - \bar{f}_j, \quad \text{for } j = n, \dots, 1.\end{aligned}$$

Therefore, the exact expression of the state range can be given as follows:

$$[\underline{s}_j, \bar{s}_j] = \begin{cases} [0, 0], & \text{for } j = 1, \\ [\underline{s}_j^B, \bar{s}_j^B] \cap [\underline{s}_j^F, \bar{s}_j^F] & \text{for } j = 2, \dots, n, \\ [l^k, u^k] & \text{for } j = n + 1. \end{cases} \quad (5.0.2)$$

If any $[\underline{s}_j, \bar{s}_j]$ is empty, then $P_\mu(\alpha, \beta)$ has no feasible solution. In general, the state space of dynamic programming can be significantly reduced by the formulas above.

Although the objective function is assumed to be integer-valued in the algorithm, we can also handle cases with a rational objective function by multiplying a suitable number.

We now present the proposed convergent surrogate dual search algorithm as follows.

5.1. Algorithm for convergent surrogate dual search

Algorithm 5.1.1. (*Convergent surrogate dual search and objective level cut*)

Step 0 (*Initialization*). Set α^0 equal to $\min_{x \in X} f(x)$ and set β equal to the objective value of an incumbent solution x_0 generated by some heuristic method when possible. When no feasible solution is available, set β equal to $\max_{x \in X} f(x)$. Choose any $\mu_0 \in \mathbf{R}_m^+$. Let $T^0 = \emptyset$ and $k = 0$.

Step 1 (*Surrogate relaxation*). Solve the surrogate relaxation problem with objective level cut $(P_{\mu_k}(\alpha^k, \beta))$. If the solution x^k satisfies $g(x^k) \leq b$, stop and x^k is an optimal solution to (P) . Otherwise, go to Step 2.

Step 2 (*Updating lower bound*). Set $\alpha^{k+1} = f(x^k)$ if $f(x^k) > \alpha^k$ and set $\alpha^{k+1} = \alpha^k$ if $f(x^k) = \alpha^k$.

Step 3 (*Updating multiplier*). Set $T^{k+1} = T^k \cup x^k$. Solve the following linear program,

$$\begin{aligned}
 (LP_k) \quad & \max_{(\gamma, \mu)} \gamma \\
 & \text{s.t. } \gamma \leq \mu^T(g(x) - b), \forall x \in T^{k+1}, \\
 & \mu \in \Lambda,
 \end{aligned}$$

where $\Lambda = \{\mu \in \mathbf{R}_+^m \mid e^T \mu \leq 1\}$ and $e = (1, \dots, 1)^T$, and obtain an optimal solution (γ^k, μ_k) . If $\gamma^k \leq 0$, go to Step 4. Otherwise, set $\mu_{k+1} = \mu_k$ and $k = k + 1$, go to Step 1.

Step 4 (*Exhausting the solutions at the level of the current lower bound*). Find out all solutions with their objective $f(x) = \alpha^{k+1}$ under constraint $\mu_k^T(g(x) - b) \leq 0$. If any one solution is feasible to the primal problem, stop and the optimal solution is found. Otherwise, set $\alpha^{k+1} = \alpha^k + 1$, $T^{k+1} = \emptyset$ and $k = k + 1$, and go to Step 1 with any $\mu_k \in \mathbf{R}_m^+$.

Theorem 5.1.1. *When the primal problem (P) is feasible, Algorithm 5.1.1 finds an optimal solution of (P) in a finite number of iterations.*

Proof. From Step 2 of the algorithm, $\alpha^k \geq f(x^i)$, for all $i = 1, \dots, k-1$, and the equality is achieved at least one of them. When $\alpha^k \leq v(P) \leq \beta$ is satisfied, $v(P_{\mu^k}(\alpha^k, \beta)) = f(x^k)$ is always a lower bound of $v(P)$. Thus, $\alpha^k \leq v(P) \leq \beta$ always holds in the whole solution process. When the algorithm stops at either Step 1 or Step 4, the strong duality holds and the primal problem is solved.

From Step 2 of the algorithm, $\{\alpha^k\}$ is a nondecreasing sequence. We only need to prove that $\{\alpha^k\}$ cannot repeat at any level infinite times. Then the condition $v(P) - \delta < \alpha^k \leq v(P)$ will be satisfied in a finite number of iterations, leading to the identification of an optimal solution to (P) and the termination of the algorithm based on Theorem 5.0.1.

Condition $\gamma^k > 0$ in Step 3 implies

$$\min_{x^i \in T^{k+1}} (\mu^{k+1})^T (g(x^i) - b) = \gamma^k > 0,$$

which further implies the infeasibility of all $x^i \in T^{k+1}$ in $(P_{\mu^{k+1}}(\alpha^{k+1}, \beta))$. Thus, no x_k found in the previous iteration will appear again as a solution to the surrogate constraint formulation in later iterations.

Entering Step 4 will result in an identification of an optimal solution to (P) or $\alpha^{k+1} = \alpha^k + 1$. Thus, no x satisfying $f(x) = \alpha^k$ will appear again as the solution to the surrogate constraint formulation. \square

5.2. Solution schemes for $(P_{\mu}(\alpha^k, \beta))$ and

$$f(x) = \alpha^k$$

The key step in Algorithm 5.1.1 is to solve the following doubly constrained surrogate constraint formulation with objective level cut:

$$\begin{aligned} (P_{\mu}(\alpha, \beta)) \quad & \min f(x) \\ & \text{s.t. } \mu^T (g(x) - b) \leq 0 \\ & \alpha \leq f(x) \leq \beta \\ & x \in X. \end{aligned}$$

Note that constraint $\mu^T(g(x)-b) \leq 0$ may not be integer valued, as surrogate multiplier vector μ is in general real-valued, which may cause large number of state grids when using dynamic programming. One feature of $(P_\mu(\alpha, \beta))$ is that the one constraint and the objective function are of the same functional form.

Recognizing the above characteristics of problem $P_\mu(\alpha, \beta)$, we are able to derive an efficient solution scheme, fortunately, by considering the following singly constrained separable integer programming problem,

$$\begin{aligned} (P_\mu^g(\alpha, \beta)) \quad & \min \mu^T g(x) \\ & \text{s.t. } \alpha \leq f(x) \leq \beta \\ & x \in X. \end{aligned}$$

Note that the assumption that $f(x)$ is integer valued facilitates a good control of the number of state grids resulted from the single constraint in $(P_\mu^g(\alpha, \beta))$.

We adopt forward dynamic programming to solve $(P_\mu^g(\alpha, \beta))$. More specifically, we apply forward dynamic programming to find solutions corresponding to $f(x) = \alpha, \alpha + 1, \dots$, successively. If the solution corresponding to $f(x) = \alpha$ in $(P_\mu^g(\alpha, \beta))$ satisfies $\mu^T(g(x) - b) \leq 0$, this solution is also optimal to $(P_\mu(\alpha, \beta))$. Otherwise, there is no solution of $(P_\mu(\alpha, \beta))$ satisfying $\mu^T(g(x) - b) \leq 0$ with $f(x) = \alpha$, the lower bound of $f(x)$ can be raised to $\alpha + 1$, and we consider next the solution of $(P_\mu^g(\alpha, \beta))$ corresponding to $f(x) = \alpha + 1$. The process continues until we first find a solution $(P_\mu^g(\alpha, \beta))$ that satisfies $\mu^T(g(x) - b) \leq 0$, which yields the solution of $(P_\mu(\alpha, \beta))$.

Example 5.2.1.

$$\begin{aligned} \min f(x) &= 2x_1^2 - 3x_2^3 + 5x_3 \\ \text{s.t. } g(x) &= -\frac{1}{2}x_1 + 2x_2 + x_3^2 \leq 1, \\ -8 &\leq f(x) = 2x_1^2 - 3x_2^3 + 5x_3 \leq 10 \\ x &\in X = \{-1, 0, 1\}^3. \end{aligned}$$

We now show how to identify the optimal solution of the above problem by solving the following singly constrained integer programming problem,

$$(D) \quad \begin{aligned} \min g(x) &= -\frac{1}{2}x_1 + 2x_2 + x_3^2 \\ \text{s.t. } -8 &\leq f(x) = 2x_1^2 - 3x_2^3 + 5x_3 \leq 10 \\ x &\in X = \{-1, 0, 1\}^3. \end{aligned}$$

The range of the state space induced by constraint $-8 \leq f(x) \leq 10$ can be determined as: $FS_1 = [0, 0]$, $FS_2 = [0, 2]$, $FS_3 = [-3, 5]$ and $FS_4 = [-8, 10]$.

Evidenced from Table 5.1 that presents the results of using forward dynamic programming, the minimum s_4 value with $t_4(s_4)$ value less than 1 is -5 . Thus, the optimal solution to this example is $\{x_1 = 0, x_2 = 0, x_3 = -1\}$.

As the objective function in (D) is originally the constraint in the example bounded above by 1, we can make use of this to cut infeasible states. The range of the objective function of problem (D) can be determined as $GS_1 = [0, 0]$, $GS_2 = [-0.5, 0.5]$, $GS_3 = [-2.5, 1]$ and $GS_4 = [-2.5, 1]$. When we perform forward dynamic programming, if the value of the minimum cost-to-accumulate function of a state, $\hat{t}_{k+1}(s_{k+1})$, is outside the feasible range of the objective value for the given stage, the corresponding decision will be blocked from further consideration.

For example, at the second stage with $s_3 = -3$, the value of the cost-to-accumulate, $\hat{t}_3(s_3)$, is 2, which is larger than the upper bound of GS_3 . Then states derived from it will not be considered in next stage. The whole process of the forward dynamic programming is shown in Table 5.2.

Problem $(P_\mu^g(\alpha, \beta))$ will be solved during the solution process many times for different μ_k , α_k and β_k . Note that α_k is nondecreasing and β_k is nonincreasing. Furthermore, when some parts of the table are removed due to violating $\mu_k^T(g(x) - b) \leq 0$ for any μ , these deleted parts must violate one constraint and they should not be considered again in the later iterations. Therefore, the size of the dynamic programming table will be monotonically decreasing.

Table 5.1: Solution process for problem D using forward dynamic programming

s_2	$x_1^*(s_2)$	$\hat{t}_2(s_2)$	s_3	$x_2^*(s_3)$	$\hat{t}_3(s_3)$	s_4	$x_3^*(s_4)$	$\hat{t}_4(s_4)$
2	1	-0.5	5	-1	-2.5	10	1	-1.5
0	0	0	3	-1	-2	8	1	-1
			2	0	-0.5	7	1	0.5
			0	0	0	5	1	1
			-1	1	1.5	4	1	2.5
			-3	1	2	3	0	-2
						2	0	-0.5
						0	0	0
						-1	0	1.5
						-2	-1	-1
						-3	-1	0.5
						-5	-1	1
						-6	-1	2.5
						-8	-1	3

We now describe how to find out all solutions with their objective $f(x) = \alpha^{k+1}$ under constraint $\mu_k^T(g(x) - b) \leq 0$. Similar to our solution scheme for $(P_\mu^g(\alpha, \beta))$, we consider a dynamic programming table induced by objective function $\mu_k^T g(x)$ and constraint $f(x) = \alpha^{k+1}$. The condition $\mu_k^T(g(x) - b) \leq 0$ can be used to reduce the size of the dynamic programming table before performing an exhausting search for solutions of $f(x) = \alpha^{k+1}$. We remark that this problem can be also solved by rank listing all solutions from the one that achieves the minimum of $\mu_k^T g(x)$ to the maximum one that satisfies $\mu_k^T g(x) \leq \mu_k^T b$.

To illustrate how the algorithm works, we consider the following example with 5 variables and 3 constraints.

Table 5.2: Solution process for problem D using forward dynamic programming incorporated with surrogate constraint.

s_2	$x_1^*(s_2)$	$\hat{t}_2(s_2)$	s_3	$x_2^*(s_3)$	$\hat{t}_3(s_3)$	s_4	$x_3^*(s_4)$	$\hat{t}_4(s_4)$
2	1	-0.5	5	-1	-2.5	10	1	-1.5
0	0	0	3	-1	-2	8	1	-1
			2	0	-0.5	7	1	0.5
			0	0	0	5	1	1
			-1	1	1.5/ ∞	4	1	2.5/ ∞
			-3	1	2/ ∞	3	0	-2
						2	0	-0.5
						0	0	0
						-1	0	1.5/ ∞
						-2	-1	-1
						-3	-1	0.5
						-5	-1	1
						-6	-1	2.5/ ∞
						-8	-1	3/ ∞

Example 5.2.2.

$$\begin{aligned}
& \min -2x_1 - 3x_1^2 + x_1^3 + 8x_2 - 7x_2^2 - 5x_3 - 3x_3^2 + 2x_4 + 4x_4^2 - 4x_5 - 7x_5^2 \\
& s.t. \ 6x_1 + 7x_1^2 + 4x_2 + 4x_2^2 + x_2^3 - 8x_3 - 7x_3^2 - 7x_4 + 2x_4^2 - 5x_5 + 2x_5^2 \leq -8, \\
& \quad 8x_1 - 5x_1^2 + 4x_2 - 7x_2^2 + x_2^3 - 4x_3 + 8x_3^2 + 7x_4 - 6x_4^2 - 2x_5 - 7x_5^2 \leq 0, \\
& \quad -x_1 - 3x_1^2 - 2x_2 + 2x_2^2 + x_2^3 - 2x_3 + 8x_3^2 - 5x_4 - 3x_4^2 + 5x_5 - 7x_5^2 \leq 3, \\
& \quad x \in X = \{x \in \mathbf{z}^5 \mid 1 \leq x_i \leq 5, i = 1, 2, 3, 4, 5\}.
\end{aligned}$$

Initial values of α^0 and β are calculated as -432 and 300 , respectively. Then the initial interval of objective cut is $[-432, 300]$. The algorithm terminates at iteration 6 when solving problem $P_\mu(-307, 300)$ and finds an optimal solution. At iteration 5, μ^k is updated to 0 and we then search all the solutions of the

surrogate constraint formulation with objective value equal to -307 . As no feasible solution to the original problem is found, we increase the lower bound by 1. Then we get the solution to $P_\mu(-306, 300)$ with value -304 , which is also feasible to the original problem. Table 5.3 summaries the iteration process of the algorithm. The real-valued μ^k is rounded off with 3 decimal numbers in the table.

Table 5.3: Iteration dynamic programming process in Example 5.3.1

Iteration	μ^k	x^*	$f(x^*)$	α^k	β
0	$(1, 0, 0)^T$			-432	300
1	$(0, 0, 1)^T$	$(2, 4, 5, 1, 5)$	-377	-377	300
2	$(0.262, 0, 0.738)^T$	$(2, 5, 1, 1, 5)$	-340	-340	300
3	$(0.070, 0, 0.930)^T$	$(2, 2, 5, 1, 5)$	-309	-309	300
4	$(0.163, 0, 0.837)^T$	$(3, 4, 4, 3, 5)$	-307	-307	300
5	$(0, 0, 0)^T$	$(3, 2, 5, 1, 5)$	-307	-306	300
6	$(1, 0, 0)^T$	$(2, 3, 4, 1, 5)$	-304		

5.3. Computational Results and Analysis

We report in this section the computational results in testing *Algorithm 5.1.1* for order-3 polynomial integer problems.

$$f_j(x_j) = \sum_{k=1}^3 c_{jk} x_j^k, \quad j = 1, \dots, n,$$

$$g_{ij}(x_j) = \sum_{k=1}^3 a_{ijk} x_j^k, \quad j = 1, \dots, m, j = 1, \dots, n.$$

Coefficients c_{jk} are set as integers with $c_{j1} \in [-20, 20]$, $c_{j2} \in [-10, 10]$, and $c_{j3} \in [-5, 5]$. Coefficients a_{ijk} are real valued numbers with $a_{ij1} \in [-20, 20]$, $a_{ij2} \in [-10, 10]$, and $a_{ij3} \in [-5, 5]$. All the coefficients are generated uniformly

and independently. The decision variables are bounded with

$$X_j = \{x_i \in \mathbf{Z} | 1 \leq x_j \leq 5\}, \quad j = 1, \dots, n.$$

Let $0 < r_i < 1, i = 1, \dots, m$. The right-hand side b_i 's take the following form,

$$b_i = \underline{g}_i + r_i(\bar{g}_i - \underline{g}_i), \quad i = 1, \dots, m,$$

where $\underline{g}_i = \min_{x \in X} g_i(x)$, $\bar{g}_i = \max_{x \in X} g_i(x)$, and the ratio r_i is used to control the size of the feasible region of the problem and the difficulty level. The experiments show that the smaller r_i is, the more cpu time is needed to identify the optimal solution. In our experiments, r_i 's are randomly generated between 0.5 and 0.7. A similar rule of determining the right-hand side was used in generating problems in Bretthauer and Shetty [5][6].

Table 5.4: Numerical results for third degree polynomial integer programming

n	m	Average No. of iterations	Average CPU times
50	20	1	0.03
50	30	4	1.60
100	40	2	0.16
100	50	3	1.07

The algorithm is coded by Fortran 90 and runs on a Dell PC with 1G ram and 2.8GHZ CPU. Table 5.4 reports sets of test problems with different number of variables and different number of constraints.

Table 5.5 compares the convergent Lagrangian dual search and objective level cut method[32] with the convergent surrogate dual search and objective level cut method for a special set of test problems with $n = 30$ and $m = 20$, it demonstrates that the surrogate dual outperforms Lagrangian dual in both dual value and in CPU time.

Table 5.5: Comparison with Lagrangian dual search method($n = 30, m = 20$)

Problem	Surrogate method		Lagrangian method	
	Initial dual value	CPU time(s)	Initial dual value	CPU time(s)
1	-5115	0.01	-5115	0.01
2	-5501	0.15	-5504	1.81
3	-5355	1.46	-5458	7.25
4	-5371	0.10	-5378	4.54
5	-6882	0.03	-6892	4.01
6	-5511	0.02	-5511	0.12
7	-7508	3.56	-7520	89.37
8	-7083	0.16	-7089	6.98
9	-5618	0.06	-5641	2.61
	Average dual value	Average time(s)	Average dual value	Average time(s)
	-5993	0.61	-6012	12.96

Table 5.6 compares convergent Lagrangian dual search and objective level cut method[32] with the convergent surrogate dual search and objective level cut method for cases where n is set as 50, in which 4 sets of test results are reported. Table 5.6 shows that the average CPU time of the convergent surrogate

Table 5.6: Comparison with Lagrangian dual search method

n	m	Surrogate method		Lagrangian method	
		Average number of iteration	Average CPU time(s)	Average number of iteration	Average CPU time(s)
50	20	2	0.03	3	9.41
50	30	4	1.60	7	30.13
50	40	3	4.71	4	11.99
50	50	3	0.37	9	13.69

dual search and objective level cut method is much less than the convergent Lagrangian dual search and objective level cut method[32].

CHAPTER 6

CONCLUSIONS

The surrogate constraint dual method is much less than the convergent Lagrangian dual search and objective level cut method[32].

Although the optimal values of the primal problem, the surrogate dual values of the surrogate constraint method are much less than the convergent Lagrangian dual search and objective level cut method[32].

End of chapter.

CHAPTER 6

CONCLUSIONS

The surrogate constraint dual method provides an alternative dual scheme for deriving tighter lower bound for multiply constrained separable integer programming problems. The surrogate relaxation, (P_μ) , differs from the primal problem, (P) , only in the feasible region. In general, the feasible region of the surrogate relaxation enlarges the feasible region of the primal problem. If this enlarged feasible region contains an infeasible solution that has an objective value less than the optimal value of (P) , then the surrogate relaxation, (P_μ) , will fail to identify an optimal solution of the primal problem, (P) , while searching for the minimum in this enlarged feasible region.

Although the optimal point of the primal problem does not attain the minimum of the surrogate relaxation when the duality gap exists, it is always hidden in the ranking order of the minimum solutions. Invoking the concept of the k th minimum path in networks, we can devise a solution scheme to pin point the optimal solution of (P) by generating successively the ranking order of the minimum solutions of a surrogate relaxation and identifying the one which first satisfies the primal feasibility. By proposing a new graph formulation of distance confined path problem, we have made substantial revisions to the traditional k th shortest path algorithms and translate them into the corresponding operations in the dynamic programming table. First, as the lower bound information dictates that the optimal value must be no less, we seek the optimality from a suitable mid-

dle point of the ranking order, thus removing unnecessary computational efforts starting from the bottom. Second, a shrinking range of the objective value allows us to reduce gradually the dynamic programming table in the solution process by removing non-promising “nodes” and “arcs” from the “graph”. This prominent feature helps speeding up the convergence. We emphasize that, as the size of the state space affects significantly the performance of dynamic programming, our first convergent surrogate dual search solution algorithm currently does not update the surrogate multipliers in order to avoid non-integer multipliers.

We impose bounds on the objective function of (P) in our second convergent surrogate dual search solution algorithm. A range reduction of the objective value forces the infeasible solutions of (P) which are better-performing gradually out of the feasible region of the surrogate relaxation, making the algorithm to converge to the optimal solution of the primal problem. Recognizing some kind of complementary positions of the objective function and the surrogate constraint in the resulting doubly constrained formulation, we switch the positions between them to benefit the algorithm from both the resulting singly constrained formulation and the surrogate dual search.

As the surrogate dual search offers a tighter bound than the more popular Lagrangian dual, our preliminary numerical experiments have demonstrated promising computational results.

□ End of chapter.

BIBLIOGRAPHY

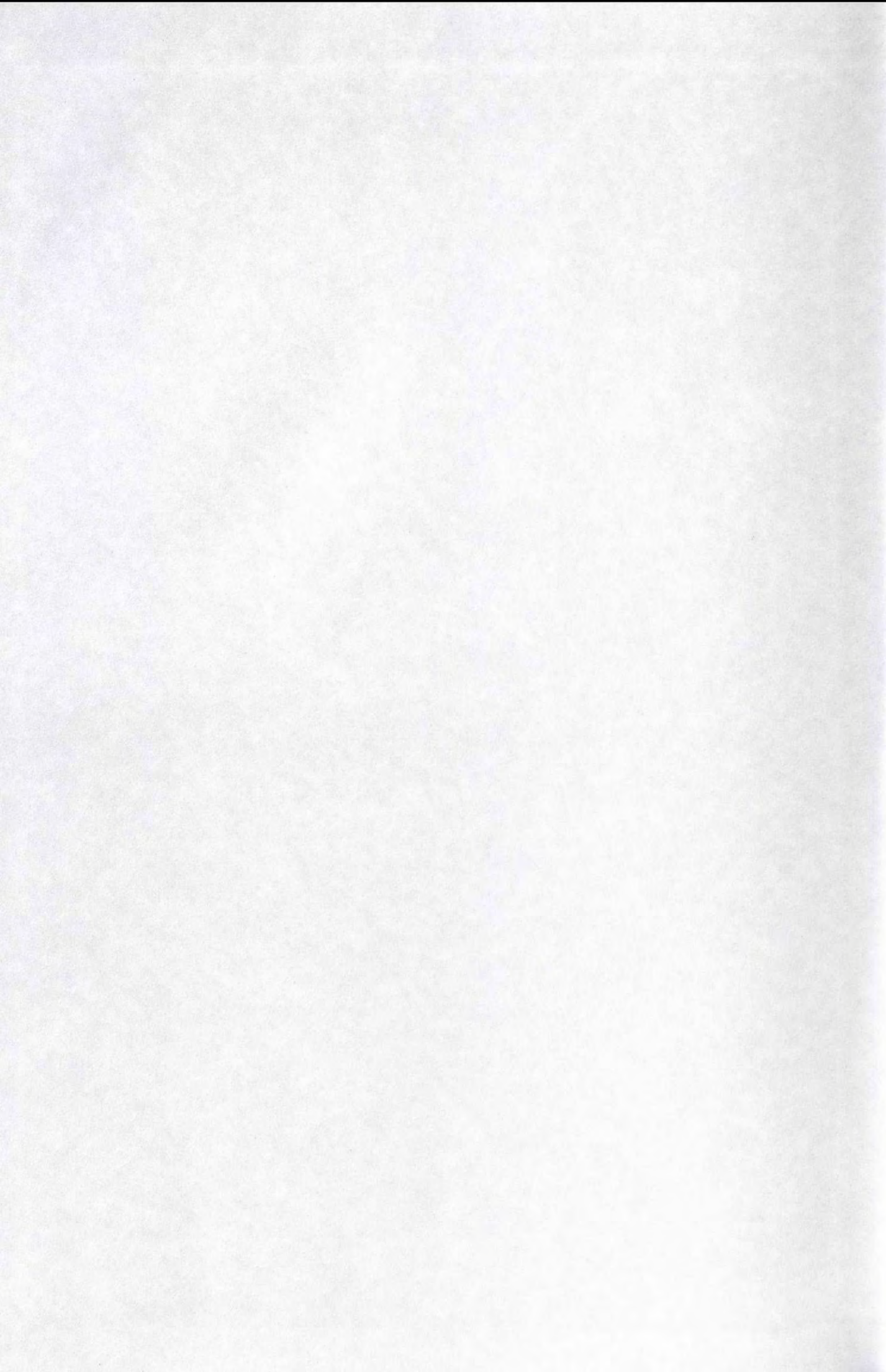
- [1] Bell, D. E. and J. F. Shapiro, A convergent duality theory for integer programming, *Operations Research* 25, 419-434, 1977.
- [2] Benson, H. P. and Erenguc, S. S., An algorithm for concave integer minimization over a polyhedron, *Naval Research Logistics* 37, 515-525, 1990.
- [3] Bertsekas, D. P. and J. Tsitsiklis, Neuro-Dynamic Programming, *Athena Scientific* 1996.
- [4] Bretthauer, K. M., Cabot, A.V. and Venkataramanan, M. A., An algorithm and new penalties for concave integer minimization over a polyhedron, *Naval Research Logistics* 41, 435-454, 1994.
- [5] Bretthauer, K. M. and B. Shetty, The nonlinear resource allocation problem, *Operations Research* 43, 670-683, 1995.
- [6] Bretthauer, K. M. and B. Shetty, A pegging algorithm for the nonlinear resource allocation problem, *Computers & Operations Research* 29, 505-527, 2002.
- [7] Byers, T. H. and M. S. Waterman, Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic programming, *Operations Research* 32, 1381-1384, 1984.
- [8] Chern, M. S., On the computational complexity of reliability redundancy allocation in a series system, *Operations Research Letters* 11, 309-315, 1992.

- [9] Dai, Y., H. Imai, K. Iwano and N. Katoh, How to treat delete requests in semi-online problems, in Proceedings of the 4th International Symposium, Algorithm and Computation, *Lecture Notes in Computer Science 762*, Springer-Verlag, New York, 48-57, 1993.
- [10] Eppstein, D., Finding the k shortest paths, *Siam Journals on Computing* 28(2), 652-673, 1999.
- [11] Fisher, M. L., The Lagrangian relaxation method for solving integer programming problems, *Management Science* 27, 1-18, 1981.
- [12] Fisher, M. L. and Sharpiro, J. F. Constructive duality in integer programming, *SIAM Journal on Applied Mathematics* 27, 31-52, 1974.
- [13] Geoffrion, A. M., Lagrangian relaxation for integer programming problems, *Mathematics Program Study* 2, 82-114, 1974.
- [14] Hadjiconstantinou, E. and N. Christofides An efficient implementation of an algorithm for finding K shortest simple paths, *Networks* 34(2), 88-101, 1999.
- [15] Haurie, A. and P. L'Ecuyer, Approximation and bounds in discrete event dynamic programming, *IEEE Transactions on Automatic control* 31, 227-235, 1986.
- [16] Hochbaum, D. S., A nonlinear knapsack problem, *Operations Research Letters* 17, 103-110, 1995.
- [17] Horst, R. and Thoai, N. V., An integer concave minimization approach for the minimum concave cost capacitated flow problem on networks, *OR Spektrum* 20, 47-53, 1998.
- [18] Horst, R. and Tuy, H., Global optimization: Deterministic Approaches, Springer, Berlin, Heidelberg, New York, 1993.

-
- [19] Ibaraki, T. and N. Katoh, Resource Allocation Problems: Algorithmic Approaches, *MIT Press*, Cambridge, Massachusetts, 1988.
- [20] Jacobson, D. and D. Mayne, Differential Dynamic Programming *Elsevier Science Publishing*, New York, 1970.
- [21] Johnson, S. A., J. R. Stedinger and C. A. Shoemaker, Numerical solution of continuous-state dynamic programs using linear and spline interpolation, *Operations Research* 41, 484, 1993.
- [22] Katoh, N., T. Ibaraki and H. Mine, Efficient algorithm for K-shortest simple paths, *Networks* 12(4), 411-427, 1982.
- [23] Kellerer, H., U. Pferschy and D. Pisinger, Knapsack Problems, *Springer Press*, Berlin, Heidelberg, 2004.
- [24] Larson, R. E., Dynamic programming with reduced computational requirements, *IEEE Transactions on Automatic Control* 10, 135-143, 1965.
- [25] Larson, R. E. and Korsak, A. J., A dynamic programming successive approximations technique with convergence proofs, *Automatica* 6, 245-252, 1970.
- [26] Lawlaer, E. L., A procedure for computing the k best solutions to discrete optimizations and its application to the shortest path problem, *Management Science* 18, 401-405, 1972.
- [27] Li, D., Zero duality gap in integer programming: P-norm surrogate constraint method, *Operations Research Letters* 25(2), 89-96, 1999.
- [28] Li, D. and X. L. Sun, Success guarantee of dual search in integer programming: p -th power Lagrangian method, *Journal of Global Optimization* 18, 235-254, 2000.

- [29] Li, D., X. L. Sun and F. L. Wang, A convergent Lagrangian and contour-cut method for nonlinear integer programming with a quadratic objective function, *SIAM Journal on Optimization* 17, 372-400, 2006.
- [30] Li, D. and X. L. Sun, Nonlinear Integer Programming, *Springer Press*, New York, 2006.
- [31] Li, D., X. L. Sun, J. Wang and K. McKinnon, Convergent Lagrangian and domain cut method for nonlinear knapsack problems, *Computational Optimization and Applications* 42, 67-104, 2009.
- [32] Li, D., J. Wang and X. L. Sun, Computing exact solution to nonlinear integer programming: Convergent Lagrangian and objective level cut method *Journal of Global Optimization* 39, 127-154, 2007.
- [33] Li, D. and D. J. White, P -th power Lagrangian method for integer programming, *Annals of Operations Research* 98, 151-170, 2000.
- [34] Liao, L. Z. and C. A. Shoemaker, Convergence in unconstrained discrete-time differential dynamic programming, *IEEE Transactions on Automatic Control* 36, 692-706, 1991.
- [35] Luus, R., Iterative dynamic programming: From curiosity to a practical optimization procedure, *Control Intelligent Systems* 26, 1-8, 1998.
- [36] Marsten, R. E. and T. L. Morin, A hybrid approach to discrete mathematical programming, *Mathematical Programming* 14, 21-40, 1978.
- [37] Mayne, D, A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems, *International Journal of Control* 3, 85-95, 1966.
- [38] Nemhauser, G. P. and L. A. Wolsey, *Integer and Combinational Optimization* Wiley, New York, 1998.

-
- [39] Shapiro, J. F., A survey of lagrangian techniques for discrete optimization, *Annals of Discrete Mathematics* 5, 113-138, 1979.
- [40] Sun, X. L. and Li, D., Asymptotic strong duality for bounded integer programming: a logarithmic-exponential dual formulation, *Mathematics of Operations Research* 25, 625-644, 2000.
- [41] Yakowite, S. and B. Rutherford, Computational aspects of discrete-time optimal control, *Journal of Applied of Mathematics and Computing* 15, 29-45, 1970.
- [42] Yen, J. Y., Finding the k shortest loopless paths in a network, *Management Science* 17, 712-716, 1971.



CUHK Libraries



004660047