Convergent Surrogate-constraint Dynamic Programming

WANG Qing

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Philosophy

in

Systems Engineering and Engineering Management

August 2006

Acknowledgement

I would like to express my sincere thanks to my supervisor, Professor Duan Li, for his precious advice and patient guidance throughout my studies and in writing this thesis.

I am also thankful to Professor Lizhi Liao for serving as the External Examiner of my thesis and to Professor XunYu Zhou and Professor Youyi Feng for serving as members of my thesis committee.

I am grateful to all my fellow students, specially to Jianjun Gao and Chunli Liu, for valuable discussion and sharing.

Finally, I would like to thank my parents for their constant love and support. I am presenting this thesis with my thanks to the department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, where I've spent six years learn, live, and grow.

Abstract

Dynamic programming, one of the powerful solution methodologies for separable optimization problems, suffers heavily from the notorious "curse of dimensionality", which prevents a direct application of dynamic programming when a large number of constraints are present. This thesis studies multiply constrained nonlinear separable integer programming problems and develops two convergent dual search algorithms for the surrogate constraint formulation of the primal problem in the sense that an exact solution of the primal problem is identified. Combined with these two convergent dual search algorithms with dynamic programming, two novel solution schemes termed *convergent surrogate constraint dynamic programming* are proposed. Numerical testing problems demonstrate some promising computational results of the two new schemes.

Keywords: separable integer programming, dynamic programming, surrogate constraint method, objective level cut, domain cut, convergent dual search.

摘要

动态规划，作为解可分离优化问题的有用解法之一，受到所谓“维数祸害”的强烈限制，使其无法直接求解约束式数量多的问题。本论文研究了多维可分离非线性整数规划问题，并提出了基于原问题迭代约束式上的两种对偶搜寻算法。综合此两种对偶搜寻算法所提出的动态规划方法被称作收敛性迭代约束动态规划。仿真测试给出了一些证实这两种新算法效用的数据。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis studies the following general class of multiply constrained separable integer programming problems:

$$
\begin{aligned}
(P) \qquad & \min\ f(x) = \sum_{j=1}^{n} f_j(x_j) \\
& \text{s.t. } g_i(x) = \sum_{j=1}^{n} g_{ij}(x_j) \le b_i,\ i = 1, \ldots, m, \\
& x \in X = X_1 \times X_2 \times \cdots \times X_n,
\end{aligned}
$$

where f_j and g_{ij}'s are defined on $\mathbb{R}$, and all X_j's are finite integer sets in $\mathbb{R}$. Let $g(x) = (g_1(x), g_2(x), \ldots, g_m(x))^T$ and $b = (b_1, b_2, \ldots, b_m)^T$. Problem (P) covers very general situations of nonlinear integer programming problems as no additional property such as convexity, concavity, monotonicity or differentiability is assumed in (P).

Without doubt, dynamic programming pioneered by Richard Bellman in 1950's is definitely one of the powerful solution methodologies for separable optimization problems by invoking the decomposition scheme based on the fundamental principle of optimality. Dynamic programming, however, suffers heavily from the notorious "curse of dimensionality", which prevents a direct application of dynamic programming when a large number of constraints are present.

1.1 Literature survey

The past four decades have witnessed tremendous efforts in the literature in alleviating the "curse of dimensionality" in dynamic programming. We first review some promising research results reported in the previous years before we propose ours.

Reference [14] deals with a discrete-time deterministic optimal control problem. Under the conventional dynamic programming, each state variable x_i, $i = 1, 2, ..., n$, is quantized to N_i levels. At each stage k, for each given quantized value of state vector $\mathbf{x}$, the optimal control $\mathbf{u}$ is identified and the corresponding minimum cost-to-go is obtained. Therefore, the computational time or storage requirements at each stage is a function of the total number of quantized states, which is $\Pi_{i=1}^{n} N_i$, an exponential function of n. The "curse of dimensionality" exhibits and prevents a direct application of dynamic programming to large-scale problems. A successive computational procedure is proposed in [14]. A nominal trajectory of state $\mathbf{x}$ and control $\mathbf{u}$ is specified first. One of the n state variables is selected while the others are held fixed. The optimization is performed and a new trajectory is obtained. Then, a different state variable is selected and the procedure repeats so that each of the state variables is selected at least once. Therefore, the original n-dimensional problem is transformed into a sequence of one dimensional problems, making the computational requirements increase linearly rather than exponentially as with the standard computation method. The "curse of dimensionality" is mitigated. However, this method may be trapped in a local minimum.

Reference [15] studies finite dynamic programming. Based on Bellman's principle of optimality, the following recursive relationship is established,

$$f(n,y) = \min_{d \in D((n,y))} \{r((n,y),d) \circ f(n-1, T((n,y),d))\}, \ n = 1,2,...,N,$$

with boundary condition

$$f(0,y) = K(0,y).$$

Before establishing the dynamic programming equations, the state space is partitioned into $N+1$ state spaces, $\Omega(0),...,\Omega(N)$. However, when the dimensionality M of the state space $\Omega(n)$ is high, the computational requirements in solving the dynamic programming equations would become excessive such that the "curse of dimensionality" exhibits. The first equation in the above recursive relationship becomes a minimum convolution by assuming that all maxima are attained:

$$f(n,y) = \min_{z \in Z(n,y)} \{r_n(z) \circ f(n-1, y-z)\}.$$

By studying the properties of this maximum convolution equation, [15] derives the following recurrence relation:

$$F_n \subset \{R_n \cup F_{n-1} \cup (R_n \oplus F_{n-1})\}, \ n = 1,2,...,N,$$

where F_n, F_{n-1} and R_n are sets of points of discontinuity of the functions $f(n,\cdot)$, $f(n-1,\cdot)$ and $r_n(\cdot)$ respectively. Note that these functions are all step functions. With the help of the above recurrence equation, there is no longer a need to solve the M-dimensional (state space) dynamic programming problem. Instead, we only need to focus on the imbedded state space F_n, which can be constructed recursively from R_n and F_{n-1}, and whose elements in $R_n \cup F_{n-1}$ or $R_n \oplus F_{n-1}$ can be eliminated. The solution procedure corresponds to a one-dimensional (state space) problem. Therefore, the "curse of dimensionality" is mitigated.

Reference [16] studies an optimal stochastic control problem for a class of discrete event systems based on a preventive replacement model. This system may be modelled as a Markovian decision process with a state-dependent discount factor. This problem may be tackled by discretizing the time, state, and action space, and then be solved using the standard solution procedure of a discrete Markovian decision process. However, the "curse of dimensionality" would occur with the discretization. Therefore, a different approach is adopted in [16]. Under certain assumptions, a value iteration method is adopted and approximation techniques are used to compute the value function at each iteration. A seven-step algorithm is proposed for the approximate computation of the optimal cost-to-go function.

Reference [13] develops the so-called differential dynamic programming (DDP) to overcome the curse of dimensionality. DDP is essentially a second-order method that successively improves the control sequence 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 curse of dimensionality. However, convergence issues may arise and paper [17] addresses the convergence issues of differential dynamic programming (DDP).

1.2 Research carried out in this thesis

The state-of-the-art in alleviating the curse of dimensionality is far below satisfaction. We suggest in this thesis a new way to tackle the curse of dimensionality in dynamic programming. Especially, we integrate the surrogate constraint formulation with dynamic programming and propose two convergent dual search methods to guarantee identification of an optimal solution of the primal problem.

The surrogate constraint method aggregates multiple constraints into one surrogate constraint, thus forming a relaxation of the primal problem. The singly-constrained relaxation problem resulted from applying the surrogate constraint method can be efficiently solved by dynamic programming. Solving the surrogate formulation does not always yield an optimal solution of the primal problem. In other words, there is no guarantee of the nonexistence of a duality gap between the primal and the dual formulations.

This thesis develops two convergent dual search algorithms for the surrogate constraint formulation which offer an optimal solution of the primal problem. Combined with these two convergent dual search algorithms, the proposed dynamic programming method is termed *convergent surrogate constraint dynamic programming*.

The first convergent surrogate constraint dynamic programming method incorporates a second constraint into the surrogate constraint formulation to force a lower bounding for the objective function. By raising the lower bound successively each time after the resulting bi-constraint relaxation problem is solved by dynamic programming, this solution process iterates and is guaranteed to converge to an optimal solution of the primal problem.

The second convergent surrogate constraint dynamic programming method integrates a domain cut procedure to remove certain unpromising sub-boxes out from further consideration. By reducing successively the enlarged portion in the feasible region of the surrogate constraint formulation with respect to the feasible region of the primal problem, the duality gap is forced to shrink. Dynamic programming is used to solve the singly-constrained surrogate formulation in each remaining sub-domain after the domain cut within a branch-and-bound framework.

The efficiency of the two proposed algorithms have been tested in several numerical testing problems and promising computation results have been observed.

Chapter 2

Conventional Dynamic Programming

Dynamic programming has been widely used in solving separable integer programming problems. The separability of both the objective function f and constraint functions g_i's makes dynamic programming method an ideal technique to solve (P). A key assumption for an efficient implementation of the dynamic programming method for (P) is the integrality of g_{ij}'s.

Assumption 2.1 *Function g_{ij} is integer-valued, for all $j = 1, \ldots, n$ and $i = 1, \ldots m$.*

2.1 Principle of optimality and decomposition

The cornerstone behind dynamic programming is the so-called principle of optimality which is invented by Richard Bellman in 1950's. To understand the principle of optimality, let us first examine an example of seeking the shortest path. Refer to Figure 2.1 in which A is the starting position. The number next

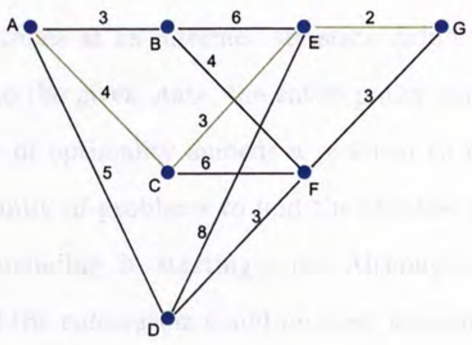


Figure 2.1: Shortest path example

to each arrow represents the distance in miles between two locations. Our goal is to reach destination G as soon as possible by finding out the shortest path from A to G. To achieve this goal, we might compare all the possible paths and then select the shortest one. In this small-size example, there are only six possible paths and this enumeration scheme is workable. But it becomes infeasible when we face a large-scale problem.

For this example problem, the shortest path from A to G passes through C and E. We claim that the path from C to G through E is the shortest distance to G starting from C. Otherwise, we may find another shorter path from C to G and combining it with the path from A to C generates an improvement of the path A-C-E-G. This is a contradiction.

The principle of optimality rests on this observation and is stated in the words of Richard Bellman, its inventor, as follows:

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.

If the remaining decisions at an intermediate state didn't constitute an optimal policy with respect to the given state, the entire policy cannot be optimal.

The principle of optimality embeds a problem to find the shortest path from A to G to a family of problems to find the shortest path to G from every intermediate point, including the starting point. Although it seems on the surface that the workload of the calculation could increase tremendously, the essence of the principle of optimality essentially leads to a powerful decomposition with a significant reduction in computation.

Let's come back to our shortest path problem and discuss a backward version of dynamic programming. We start from the last stage before reaching destination G (Stage 3). See Figure 2.2. At both E and F, there is no choice of decision. Thus the minimum distance to G from E and F (termed cost-to-go) are 2 and 3, respectively, and are recorded next to E and F, respectively.

Next we move back to the stage next to the last (Stage 2) to calculate the minimum distance to G from B, C and D. See Figure 2.3. In carrying out the calculation, only the immediate distance between stages 2 and 3 and the cost-to-go information at stage 3 are needed. The cost-to-go (minimum distance to G) is recorded in a small circle next to B, C and D, respectively, and the optimal paths going out from B, C and D are colored with yellow.

Finally, we move back to the starting point A. See Figure 2.4. In finding out the minimum distance from A to G, we only need to require knowledge of the immediate distance between A and points in stage 2 and the cost-to-go information at stage 2, since cost-to-go at stage 2 aggregates all essential

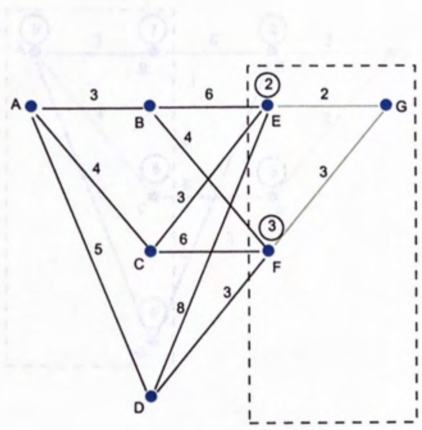


Figure 2.2: Cost-to-go at the last stage

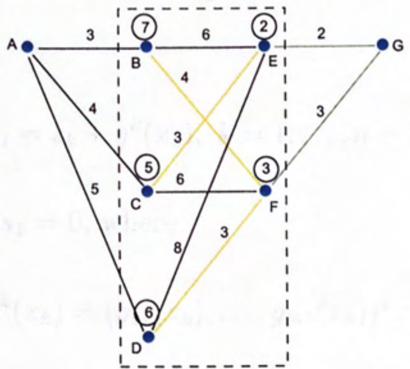


Figure 2.3: Cost-to-go at the stage next to the last

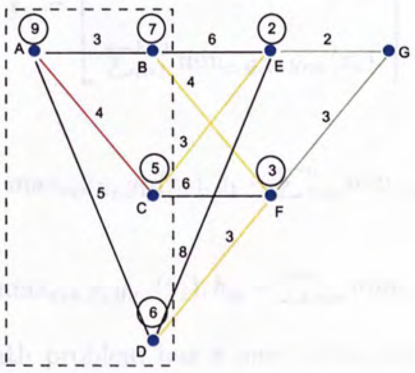


Figure 2.4: Cost-to-go at A

information related to the shortest paths from stage 2 to the destination. The optimal path going out from A is colored with red.

To apply dynamic programming to (P), we first introduce a stage variable k, $0 \le k \le n$, and a state vector at stage k, $s_k \in \mathbb{R}^m$, satisfying the following recursive equation:

$$s_{k+1} = s_k + g^k(x_k), \ k = 1, \ldots, n-1, \tag{2.1}$$

with an initial condition $s_1 = 0$, where

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

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 $2 \le k \le n+1$ can be confined as follows:

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

where

$$\underline{s}_k = \begin{bmatrix} \sum_{t=1}^{k-1} \min_{x_t \in X_t} g_{1t}(x_t) \\ \vdots \\ \sum_{t=1}^{k-1} \min_{x_t \in X_t} g_{mt}(x_t) \end{bmatrix}, \tag{2.2}$$

and

$$\bar{s}_k = \begin{bmatrix} \min\{\sum_{t=1}^{k-1} \max_{x_t \in X_t} g_{1t}(x_t), b_1 - \sum_{t=k}^{n} \min_{x_t \in X_t} g_{1t}(x_t)\} \\ \vdots \\ \min\{\sum_{t=1}^{k-1} \max_{x_t \in X_t} g_{mt}(x_t), b_m - \sum_{t=k}^{n} \min_{x_t \in X_t} g_{mt}(x_t)\} \end{bmatrix}. \tag{2.3}$$

The shortest path problem has a one-to-one correspondence to problem (P) studied in this thesis. The locations in the shortest path problem correspond to the possible values of the state variable and the paths at given locations correspond to the decisions for given states, the distance between the locations corresponds to the immediate cost, and the shortest distance to the destination corresponds to the optimal cost-to-go. Therefore, problem (P) can be interpreted as finding the optimal path with the minimum total cost starting from the initial state. More specifically, recursive relationship can be developed based on the principle of optimality to decompose problem (P) into a family of single-stage optimization problems. Figure 2.5 gives a visualization of this view.

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,

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

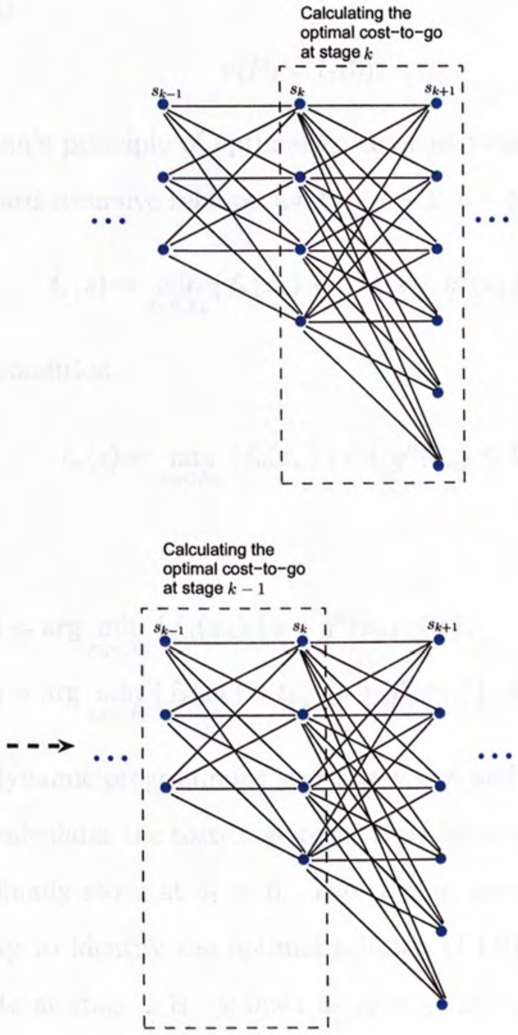

Calculating the optimal cost-to-go at stage k
s_{k-1}
s_k
s_{k+1}
Calculating the optimal cost-to-go at stage $k-1$
s_{k-1}
s_k
s_{k+1}

Figure 2.5: Recursive solution procedure for problem (P) using dynamic programming

$$\text{s.t. } s + \sum_{j=k}^{n} g^j(x_j) \le b,$$
$$x_j \in X_j,\ j = k, \dots, n.$$

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

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

Define

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

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 at stage k between $\underline{s}_k$ and $\bar{s}_k$ 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 $x_n^*(s_n^*)$.

2.3 Forward dynamic programming

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

$$\tilde{t}_k(s) = \min \sum_{j=1}^{k-1} f_j(x_j),$$

$$\text{s.t.} \quad \sum_{j=1}^{k-1} g^j(x_j) \le s,$$

$$x_j \in X_j,\ j = 1, \ldots, k-1.$$

It is obvious that

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

Based on Bellman's principle of optimality, the cost-to-accumulate function satisfies the following forward recursive relation for $k = 3, \ldots n+1$,

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

with boundary condition

$$\tilde{t}_2(s) = \min_{x_1 \in X_1} \{f_1(x_1) \mid g^1(x_1) \le s\}.$$

Define

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

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

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

The forward dynamic programming starts at $k = 2$ and moves forward, $k = 3$, $\ldots$, $n+1$. 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

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

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 2.1

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

The optimal solution is $x^* = (1, 0, -1)^T$ with $f(x^*) = -6$.

Using the formulas in (2.2) and (2.3), the feasible regions of the state vector can be found as follows for $k = 2$, 3, and 4,

$$\begin{bmatrix} -2 \\ -1 \end{bmatrix} \leq s_2 \leq \begin{bmatrix} \min\{0, -1\} \\ \min\{1, 3\} \end{bmatrix},$$
$$\begin{bmatrix} -3 \\ -2 \end{bmatrix} \leq s_3 \leq \begin{bmatrix} \min\{1, -2\} \\ \min\{2, 2\} \end{bmatrix},$$
$$\begin{bmatrix} -4 \\ -2 \end{bmatrix} \leq s_4 \leq \begin{bmatrix} \min\{2, -3\} \\ \min\{3, 2\} \end{bmatrix}.$$

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 $s_3 = (-2, -1)^T$,

Table 2.1: Solution process for Example 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	$x_3^*(s_3)/\hat{t}_3(s_3)$
$(0,0)^T$	$1/-6$	$(-2,-1)^T$	$0/-3$	$(-3,-2)^T$	$-1/-3$
		$(-2,0)^T$	$0/-3$	$(-3,-1)^T$	$-1/-3$
		$(-2,1)^T$	$0/-3$	$(-3,0)^T$	$-1/-3$
		$(-1,-1)^T$	$1/2$	$(-3,1)^T$	$-1/-3$
		$(-1,0)^T$	$1/2$	$(-3,2)^T$	$0/0$
		$(-1,1)^T$	infeasible/∞	$(-2,-2)^T$	$-1/-3$
				$(-2,-1)^T$	$-1/-3$
				$(-2,0)^T$	$-1/-3$
				$(-2,1)^T$	$-1/-3$
				$(-2,2)^T$	infeasible/∞

both $x_3 = 0$ and $x_3 = 1$ are infeasible. The optimal decision $x_3^*((-2,-1)^T)$ is found to be -1 and the corresponding $\hat{t}_3((-2,-1)^T)$ is -3. 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 $s_2 = (-2,0)^T$, comparison of $5(-1)^2+\hat{t}_3((-1,-1)^T) = \infty$, $5(0)^2+\hat{t}_3((-2,0)^T) = -3$, and $5(1)^2+\hat{t}_3((-3,1)^T) = 2$ yields $x_2^*((-2,0)^T) = 0$ and $\hat{t}_2((-2,0)^T) = -3$. 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 $x_1^*(s_1 = (0,0)^T) = 1$ and $\hat{t}_1(s_1 = (0,0)^T) = -6$. Tracing back, we find the optimal solution for the example problem: $x_1 = 1$, $x_2 = 0$ and $x_3 = -1$.

Next we examine how the forward dynamic programming is used to solve Example 2.1. Table 2.2 summarizes the solution process.

Table 2.2: Solution process for Example 2.1 using forward dynamic programming.

s_2	$x_1^*(s_2)/\tilde{t}_2(s_2)$	s_3	$x_2^*(s_3)/\tilde{t}_3(s_3)$	s_4	$x_3^*(s_4)/\tilde{t}_4(s_4)$
$(-2,-1)^T$	$-1/3$	$(-3,-2)^T$	infeasible/∞	$(-4,-2)^T$	infeasible/∞
$(-2,0)^T$	$-1/3$	$(-3,-1)^T$	infeasible/∞	$(-4,-1)^T$	infeasible/∞
$(-2,1)^T$	$1/-3$	$(-3,0)^T$	$1/8$	$(-4,0)^T$	infeasible/∞
$(-1,-1)^T$	$-1/3$	$(-3,1)^T$	$1/8$	$(-4,1)^T$	$-1/5$
$(-1,0)^T$	$-1/3$	$(-3,2)^T$	$1/2$	$(-4,2)^T$	$-1/5$
$(-1,1)^T$	$1/-3$	$(-2,-2)^T$	infeasible/∞	$(-3,-2)^T$	infeasible/∞
		$(-2,-1)^T$	$0/3$	$(-3,-1)^T$	infeasible/∞
		$(-2,0)^T$	$0/3$	$(-3,0)^T$	$0/3$
		$(-2,1)^T$	$0/-3$	$(-3,1)^T$	$-1/0$
		$(-2,2)^T$	$1/2$	$(-3,2)^T$	$-1/-6$

The solution process using forward dynamic programming starts from stage 2 and ends at stage 4. Minimizing $\tilde{t}_4$ with respect to $s_4 \leq (-3,2)^T$ finds out the optimal value of the example problem $\tilde{t}_4((-3,2)^T) = -6$. Tracing back identifies optimal solution: $x_3^* = -1$, $x_2^* = 0$ and $x_1^* = 1$.

Determining the feasible region could become a tedious task in applying dynamic programming. This difficulty can be alleviated to certain degree when the following assumption is satisfied.

Assumption 2.2 *For all $j = 1, \ldots, n$ and $i = 1, \ldots m$, function g_{ij} is integer-valued and is nonnegative for all $x_j \in X_j$.*

When Assumption 2.2 is satisfied, the range of s_k at stage k, for $k = 2, 3, \ldots, n, n+1$, can be simply determined by $[(0,\ldots,0)^T, (b_1,\ldots,b_m)^T]$.

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 2.2 holds for the transformed problem. The range of $(s_k)_i$ at stage k for $k = 2, 3, \ldots, n, n+1$ can be then given by $[0, b_i - \sum_{j \in I_i} \min_{x_j \in X_j} g_{ij}]$, where $I_i = \{j = 1, \ldots, n \mid \min_{x_j \in X_j} g_{ij} < 0\}$. 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.

2.4 Curse of dimensionality

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 multiply constrained cases of (P) when m is large.

Consider another problem with 3 variables and 5 constraints:

$$\begin{aligned}
\min\ & f(x) = 2x_1^2 + 3x_2 - 2x_3^3 \\
\text{s.t.}\ & g_1(x) = x_1^2 - x_2 - x_3 \le 0, \\
& g_2(x) = 3x_1 + x_2 + x_3 \le -1, \\
& g_3(x) = -x_1^2 - 2x_2 + 2x_3^3 \le 0, \\
& g_4(x) = -x_1^2 - x_2^2 - x_3^2 \le -1, \\
& g_5(x) = x_1^3 - 2x_2 - 3x_3 \le 3, \\
& x_i \in \{-1, 0, 1\},\ i = 1, 2, 3, 4, 5.
\end{aligned}$$

Using the formulas in (2.2) and (2.3), the feasible regions of the state vector can be found as follows for $k = 2$ and 3,

$$\begin{bmatrix} 0 \\ -3 \\ -1 \\ -1 \\ -1 \end{bmatrix} \leq s_2 \leq \begin{bmatrix} \min\{1,2\} \\ \min\{3,1\} \\ \min\{0,4\} \\ \min\{0,1\} \\ \min\{1,8\} \end{bmatrix},$$

$$\begin{bmatrix} -1 \\ -4 \\ -3 \\ -2 \\ -3 \end{bmatrix} \leq s_3 \leq \begin{bmatrix} \min\{2,1\} \\ \min\{4,0\} \\ \min\{2,2\} \\ \min\{0,0\} \\ \min\{3,6\} \end{bmatrix}.$$

If backward dynamic programming is applied, there are totally $2 \times 5 \times 2 \times 2 \times 3 = 120$ s_2 states and $3 \times 5 \times 6 \times 3 \times 7 = 1890$ s_3 states. From this calculation, we see that the number of possible states increases exponentially with respect to the number of constraints. Therefore, when m is large, the computational efforts for $x_k^*(s_k)$ and $\hat{t}_k(s_k)$ as well as the storage requirements for these calculated amounts become excessive. Generally, when we are solving a problem, our objective is to obtain the solution or the approximate solution, and to obtain it in reasonable amount of time and with reasonable utilization of computer resource. However, with dynamic programming being the solution technique, as the number, or otherwise the dimension, of the state variables increase, the computational and storage requirements grow rapidly beyond the handling of the most up-to-date computer device. This prohibits the direct application of the dynamic programming technique to problems when the dimension of the state variables is relatively high or the number of the state variables is relatively large. This phenomenon is known in literature as the "curse of dimensionality".

Dynamic programming, however, remains as an efficient solution scheme for separable integer programming problems when m is small, especially for singly constrained cases.

2.5 Singly constrained case

Consider the singly constrained case of (P):

$$\begin{aligned}(P_1) \qquad & \min \ f(x) = \sum_{j=1}^{n} f_j(x_j) \\ & \text{s.t. } \ g(x) = \sum_{j=1}^{n} g_j(x_j) \le b, \\ & \qquad x \in X = X_1 \times X_2 \times \cdots \times X_n,\end{aligned}$$

where $X_j = \{x_j \in \mathbb{Z} \mid l_j \le x_j \le u_j\}$ with l_j and u_j being integers. We assume $g_j(x_j) \ge 0$ on X_j for all $j = 1, \ldots, n$.

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

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

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

$$\begin{aligned}\hat{t}_k(s) &= \min\{f_k(x_k) + \hat{t}_{k+1}(s + g_k(x_k))\} \\ \text{s.t. } \ & s + g_k(x_k) \le b, \\ & x_k = l_k, \ldots, u_k,\end{aligned}$$

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

$$\begin{aligned}
\hat{t}_k(s) &= +\infty, \text{ for } s < 0, \ k = 1, \ldots, n, \\
\hat{t}_n(s) &= \min\{f_n(x_n) \mid s + g_n(x_n) \le b, \ x_n = l_n, l_n + 1, \ldots, u_n\}, \\
& \quad s = 0, \ldots, b.
\end{aligned}$$

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

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

The *forward* recursive equation is

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

for $k = 3, \ldots, n$, $s = 0, \ldots, b$, with boundary conditions

$$\begin{aligned}
\tilde{t}_j(s) &= +\infty, \quad \text{for } s < 0, \ j = 1, \ldots, n, \\
\tilde{t}_2(s) &= \min\{f_1(x_1) \mid g_1(x_1) \le s, \ x_1 = l_1, l_1 + 1, \ldots, u_1\}, \\
& \quad s = 0, \ldots, b.
\end{aligned}$$

The dynamic programming table has a size of $n \times (b+1)$.

Example 2.2

$$\begin{aligned} \min\ & f(x) = -2\sqrt{x_1} - 2x_2 - x_3^2 - (1/2)x_4^3 \\ \text{s.t.}\ & g(x) = 3x_1 - x_1^2 + x_2 + x_3^2 + x_4 \le 5, \\ & x \in [0,2]^4 \cap \mathbb{Z}^4. \end{aligned}$$

The optimal solution is $x^* = (0,2,1,2)^T$ with $f(x^*) = -9$.

Table 2.3 shows the process of the forward dynamic programming for this example, where $w_k(s) = s - g_k(x_k^*(s))$.

Table 2.3: Dynamic programming table for Example 2.2.

s	$\tilde{t}_2(s)/x_1^*(s)$	$\tilde{t}_3(s)/x_2^*(s)/w_2(s)$	$\tilde{t}_4(s)/x_3^*(s)/w_3(s)$	$\tilde{t}_5(s)/x_4^*(s)/w_4(s)$
0	0/0	0/0/0	0/0/0	0/0/0
1	0/0	−2/1/0	−2/0/1	−2/0/1
2	−2.8284/2	−4/2/0	−4/0/2	−4/2/0
3	−2.8284/2	−4.8284/1/2	−5/1/2	−6/2/1
4	−2.8284/2	−6.8284/2/2	−6.8284/0/4	−8/2/2
5	−2.8284/2	−6.8284/1/2	−7.8284/1/4	−9/2/3

Thus $\tilde{t}_5(5)$ is the optimal value and the optimal solution can be obtained by backtracking out through the table:

$$\begin{aligned} & s_5^* = 5, x_4^* = 2 \Longrightarrow s_5^* - g_4(x_4^*) = 3 \\ & s_4^* = 3, x_3^* = 1 \Longrightarrow s_4^* - g_3(x_3^*) = 2 \\ & s_3^* = 2, x_2^* = 2 \Longrightarrow s_3^* - g_2(x_2^*) = 0 \\ & s_2^* = 0, x_1^* = 0. \end{aligned}$$

Therefore the optimal solution is $x^* = (0,2,1,2)^T$.

Chapter 3

Surrogate Constraint Formulation

3.1 Conventional surrogate constraint formulation

The surrogate constraint formulation has been widely used in solving integer programming problems. More specifically, the surrogate constraint formulation is formed by aggregating multiple constraints into a single surrogate constraint,

$$
\begin{array}{lll}
(P_\mu) & & \min\ f(x) \\
 & & \text{s.t.}\ \ \mu^T(g(x)-b) \le 0, \\
 & & \qquad x \in X,
\end{array}
$$

where $\mu = (\mu_1, \ldots, \mu_m)^T \in \mathbb{R}_+^m$ is a vector of surrogate multipliers, $g(x) = (g_1(x), \ldots, g_m(x))^T$ and $b = (b_1, \ldots, b_m)^T$. Define $S(\mu)$ to be the feasible region of decision vectors in (P_μ),

$$S(\mu) \ = \ \{x \in X \mid \mu^T(g(x)-b) \le 0\}. \tag{3.1}$$

Clearly, compared to the feasible region of decision vectors in the primal problem,

$$S = \{x \in X \mid g(x) - b \le 0\},$$

the following relationship holds for any $\mu \in \mathbb{R}^m_+$,

$$S \subseteq S(\mu).$$

Denote by $v(Q)$ the optimal value of an optimization problem (Q). Since, for any $\mu \in \mathbb{R}^m_+$, $S(\mu)$ is an enlargement of the feasible set of the primal problem, the following weak surrogate duality is evident,

$$v(P_\mu) \le v(P), \quad \forall\, \mu \in \mathbb{R}^m_+.$$

Surrogate constraint formulation (P_μ) with a parameter μ is called a relaxation of the primal problem (P) since $v(P_\mu) \le v(P)$ holds for all possible values of μ. In other words, solving a relaxation problem offers a lower bound of the optimal value of the primal problem. The dual problem is formulated to search for an optimal parameter, μ^*, such that the duality gap of $v(P) - v(P_\mu)$ is minimized at $\mu = \mu^*$. The quality of a relaxation should be thus judged by two measures. The first measure is how easier the relaxation problem can be solved when compared with the primal problem. The second measure is how tight the lower bound can be, in other words, how small the duality gap can be reduced to.

Note that surrogate constraint formulation (P_μ) is a singly constrained separable integer programming problem which can be efficiently solved by dynamic programming.

The surrogate dual is an optimization problem in μ,

$$(D_S) \qquad \begin{aligned} &\max\ v(P_\mu) \\ &\text{s.t.}\ \ \mu \in \mathbb{R}^m_+. \end{aligned}$$

Consequently, based on the weak surrogate duality, the surrogate dual provides a best lower bound for $v(P)$ from solving (P_μ),

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

When a solution in the surrogate constraint formulation achieves an optimality in the primal, we say that strong surrogate duality holds. The strong surrogate duality rests on the feasibility of the solution to (P_{μ^*}).

Theorem 3.1 (Strong Surrogate Duality) *[9] If an x^* solves (P_{μ^*}) for a $\mu^* \in \mathbb{R}^m_+$ and x^* is feasible in (P), then x^* solves (P) and $v(D_S) = v(P)$.*

It is clear that $v(P_\mu) = v(P_{\theta\mu})$ 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 $\Lambda = \{\mu \in \mathbb{R}^m_+ \mid e^T\mu \leq 1\}$ and $e = (1, \ldots, 1)^T$.

3.2 Surrogate dual search

How to update the surrogate multipliers in order to maximize the surrogate dual is a key to success in applying the surrogate dual method. The discussion in this section is based on [9].

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

$$S(\mu) \cap X(\alpha) \neq \emptyset, \tag{3.2}$$

where $S(\mu)$ is defined by (3.1). Consider the following problem

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

We notice that (3.2) holds if and only if $v(P(\alpha,\mu)) \le 0$. Since $v(D_S^n) = \max\{v(P_\mu) \mid \mu \in \Lambda\}$, it follows that $v(D_S^n) \le \alpha$ if and only if $v(P(\alpha,\mu)) \le 0$ for all $\mu \in \Lambda$. We define now the following dual problem:

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

The above discussion leads to the following theorem.

Theorem 3.2 *[9] For given $\alpha \in \mathbb{R}$, $v(D_S^n) \le \alpha$ if and only if $v(D(\alpha)) \le 0$.*

An immediate corollary of Theorem 3.2 is as follows.

Corollary 3.2.1 *[9] The optimal surrogate dual value $v(D_S^n)$ is the minimum $\alpha \in \mathbb{R}$ such that $v(D(\alpha)) \le 0$.*

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

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

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

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

The above discussion leads to the following algorithm to solve (D_S^n).

Procedure 3.1 (Cutting Plane Procedure for (D_S^n))

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

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

Step 2 (Updating lower bound). If $f(x^k) > \alpha^{k-1}$, 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 $\beta^k \le 0$, 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 *[9] Algorithm 3.1 finds an optimal value of (D_S^n) within a finite number of iterations.*

To illustrate Procedure 3.1, consider the following example:

Example 3.1

$$\begin{aligned} \min\ & f(x) = 3x_1^2 + 2x_2^2 \qquad (3.3)\\ \text{s.t.}\ & g_1(x) = 10 - 5x_1 - 2x_2 \le 7,\\ & g_2(x) = 15 - 2x_1 - 5x_2 \le 12,\\ & x \in X = \left\{ \begin{array}{c} \text{integer} \\ 0 \ \le x_1 \ \le 1,\ 0 \le x_2 \le 2 \\ 8x_1 + 8x_2 \ge 1 \end{array} \right\}. \end{aligned}$$

The optimal solution is $x^* = (1,1)^T$ with $f(x^*) = 5$.

The iteration process of Procedure 3.1 for this example is described as follows:

Step 0. Set $\alpha^0 = -\infty$, $T^0 = \emptyset$. Choose $\mu^1 = (0.5, 0.5)^T$. Set $k = 1$.

Iteration 1

Step 1. Solve the surrogate problem

$$\begin{aligned} (P_{\mu^1}) \qquad \min\ & 3x_1^2 + 2x_2^2\\ \text{s.t.}\ & 0.5 \times (10 - 5x_1 - 2x_2) + 0.5 \times (15 - 2x_1 - 5x_2) \le 9.5,\\ & x \in X. \end{aligned}$$

We obtain $x^1 = (0,1)^T$ with $g(x^1) = (8,10)^T \not\le (7,12)^T$.

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

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

$$\begin{aligned} (LP_1) \qquad \max_{(\beta,\mu)}\ & \beta\\ \text{s.t.}\ & \beta \le \mu_1 - 2\mu_2,\\ & \mu_1 + \mu_2 \le 1,\\ & \mu_1 \ge 0,\ \mu_2 \ge 0. \end{aligned}$$

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

Iteration 2

Step 1. Solve the surrogate problem

$$
\begin{aligned}
(P_{\mu^2}) \qquad & \min\ 3x_1^2 + 2x_2^2 \\
& \text{s.t.}\ \ 1 \times (10 - 5x_1 - 2x_2) + 0 \times (15 - 2x_1 - 5x_2) \leq 7, \\
& \qquad x \in X.
\end{aligned}
$$

We obtain $x^2 = (1,0)^T$ with $g(x^2) = (5,13)^T \not\leq (7,12)^T$.

Step 2. Since $f(x^2) = 3 > \alpha^1$, set $\alpha^2 = 3$.

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

$$
\begin{aligned}
(LP_2) \qquad & \max_{(\beta,\mu)}\ \beta \\
& \text{s.t.}\ \ \beta \leq \mu_1 - 2\mu_2, \\
& \qquad \beta \leq -2\mu_1 + \mu_2, \\
& \qquad \mu_1 + \mu_2 \leq 1, \\
& \qquad \mu_1 \geq 0,\ \mu_2 \geq 0.
\end{aligned}
$$

We obtain $\beta^2 = 0$ and $\mu^2 = (0,0)^T$. Stop and the optimal surrogate dual value is $v(D_S^n) = \alpha^2 = 3$. Note that the surrogate dual value, 3, is better than the Lagrangian dual value, $2\frac{1}{3}$.

3.3 Nonlinear surrogate constraint formulation

The surrogate constraint method does not always solve the primal problem, i.e., the surrogate dual does not guarantee generation of an optimal solution of the primal problem. When $v(D_S)$ is strictly less than $v(P)$, a duality gap exists between (D_S) and (P).

As stated in [9]: 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 a point that is infeasible with respect to the major constraints of the primal problem and has a smaller objective value than $v(P)$, 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.

To illustrate this argument further, we consider Example 5.1 again. Applying the conventional surrogate constraint method to solve (3.3) yields,

$$\begin{aligned} &\min\ 3x_1^2 + 2x_2^2 \\ &\text{s.t.}\ \ \mu_1(10 - 5x_1 - 2x_2) + \mu_2(15 - 2x_1 - 5x_2) \le 7\mu_1 + 12\mu_2, \\ &\qquad x \in X = \left\{ \begin{array}{c} \text{integer} \\ 0 \le x_1 \le 1,\ 0 \le x_2 \le 2 \\ 8x_1 + 8x_2 \ge 1 \end{array} \right\}. \end{aligned} \tag{3.4}$$

It can be seen from Figure 3.1 that the surrogate constraint defines a closed half space in the $\{g_1, g_2\}$ space. For whatever value of μ chosen, the resulting closed half space always includes an infeasible solution of the primal problem. Both infeasible solutions, $(0,1)^T$ and $(1,0)^T$, in this example have objective values smaller than $v(P)$. As a result, the conventional surrogate constraint method fails to generate the optimal solution of the primal problem in this example. The resulting maximum dual value is $v(P_\mu) = 3$ with $\mu = (1,0)^T$ as been computed in Example 5.1, and a duality gap exists.

It becomes clear now that a sufficient requirement to eliminate the duality gap in the surrogate constraint method is to make the feasible region in the

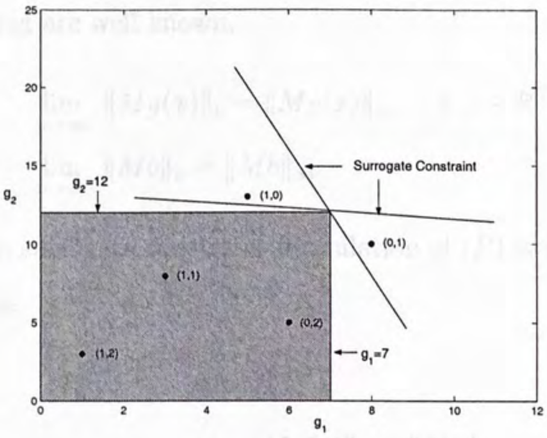


Figure 3.1: Surrogate constraints in the constraint space of Example 5.1.

constraint space, defined by a single surrogate constraint, the same as the feasible region in the primal problem. This goal can be achieved by some nonlinear surrogate constraint methods discussed in [9].

Without loss of generality, $g_i(x)$, $i = 1, 2, \ldots, m$, are assumed to be strictly positive for all $x \in X$.

Let $M = diag(\mu_1, \ldots, \mu_m)$. We define the following weighted p-norms, for a real number p with $1 \leq p < \infty$, as:

$$\|Mg(x)\|_p = \{\sum_{i=1}^{m} [\mu_i g_i(x)]^p\}^{1/p},$$

$$\|Mb\|_p = \{\sum_{i=1}^{m} [\mu_i b_i]^p\}^{1/p},$$

and the weighted ∞-norm as

$$\|Mg(x)\|_\infty = \max\{\mu_1 g_1(x), \mu_2 g_2(x), \ldots, \mu_m g_m(x)\},$$

$$\|Mb\|_\infty = \max\{\mu_1 b_1, \mu_2 b_2, \ldots, \mu_m b_m\}.$$

The following are well known,

$$\lim_{p\to\infty} \|Mg(x)\|_p = \|Mg(x)\|_\infty, \quad \forall\, x \in \mathbb{R}^n,$$
$$\lim_{p\to\infty} \|Mb\|_p = \|Mb\|_\infty.$$

The p-norm surrogate constraint formulation of (P) is now formed as follows for $1 \le p < \infty$:

$$(P_\mu^p) \qquad \min\ f(x)$$
$$\text{s.t.}\ \ \|Mg(x)\|_p \le \|Mb\|_p,$$
$$x \in X,$$

where μ satisfies the following,

$$\mu_1 b_1 = \mu_2 b_2 = \ldots = \mu_m b_m. \tag{3.5}$$

Let B be a positive real number that is defined as follows for a μ that satisfies (3.5),

$$B = \mu_i b_i, \quad i = 1, 2, \ldots, m.$$

Define $S^p(\mu)$ to be the feasible region of decision vectors in (P_μ^p),

$$S^p(\mu) = \{x \in X \mid \|Mg(x)\|_p \le \|Mb\|_p\}.$$

When x satisfies $g_i(x) \le b_i$, $i = 1, 2, \ldots, m$, x also satisfies $\|Mg(x)\|_p \le \|Mb\|_p$ for $1 \le p < \infty$. Thus, $S \subseteq S^p(\mu)$ when $1 \le p \le \infty$. Thus, when $1 \le p < \infty$, problem (P_μ^p) is a relaxation of problem (P) and we have

$$v(P_\mu^p) \le v(P), \quad \forall\, 1 \le p < \infty.$$

While (P_μ^p) still constitutes a relaxation of problem (P) even when μ does not satisfy (3.5), the p-norm surrogate constraint method confines itself to use only those μ's that satisfy (3.5), due to several important properties associated with μ satisfying (3.5), as indicated in [9].

For $\mu \in \mathbb{R}_+^m$ satisfying (3.5), let $G^p(\mu)$ denote the feasible region formed by the p-norm surrogate constraint in the g space,

$$G^p(\mu) = \{g \in \mathbb{R}_+^m \mid \|Mg\|_p \leq \|Mb\|_p\}.$$

Figure 3.2 graphically demonstrates the feasible regions in the $\{g_1, g_2\}$ space defined by the p-norm surrogate constraint for different values of p. A nice property of inclusion can be seen for $G^p(\mu)$ from Figure 3.2 and this property of inclusion is mathematically proven in the following theorem.

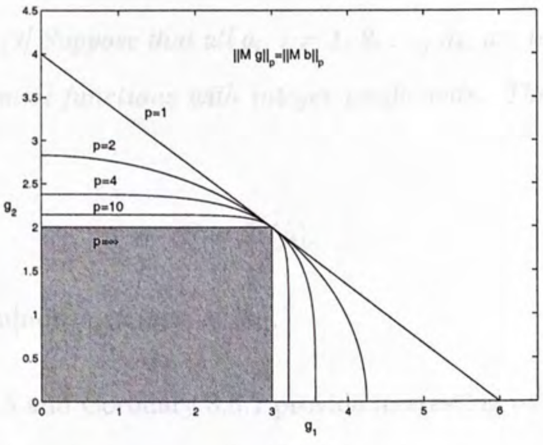


Figure 3.2: p-norm surrogate constraints in the constraint space with $\mu = (0.4, 0.6)^T$ and $b = (3, 2)^T$.

Theorem 3.4 *[9] For* $\infty > p > q$,

$$G^p(\mu) \subseteq G^q(\mu),$$

where μ satisfies (3.5).

The following theorem further establishes an equivalence between (P_μ^p) and (P) with respect to the feasible region of decision vectors.

Theorem 3.5 *[9] If μ satisfies (3.5), then there exists a finite q such that*

$$S = S^p(\mu)$$

for all $p > q$.

In general, obtaining q could be at least as difficult as solving (P) itself. For an important general class of integer programming problems, however, a lower bound of p can be easily calculated.

Corollary 3.3.1 *[9] Suppose that all g_i, $i = 1, 2, \ldots, m$, are integer-valued functions, e.g., polynomial functions with integer coefficients. Then for μ satisfying (3.5),*

$$S = S^p(\mu).$$

when $p > \ln(m)/\ln[\min_{1\le i\le m}(b_i + 1)/b_i]$.

Theorem 3.5 and Corollary 3.3.1 provide interesting results in separation. By selecting a sufficiently large p, all infeasible solutions of the primal problem will be excluded from $S^p(\mu)$. In other words, the feasible set defined by the p-norm surrogate constraint, $S^p(\mu)$, will exactly match the feasible set of the primal problem, S, when $p > q$. In summary, an appropriately selected nonlinear single surrogate constraint can be constructed by aggregating multiple major constraints of the primal problem such that a surrogate relaxation and the primal problem are exactly equivalent. This result offers a basis in achieving zero duality gap in integer programming when adopting the p-norm surrogate constraint method.

Theorem 3.6 *[9] Suppose that all g_i, i = 1, 2, ..., m, are integer-valued functions. If μ satisfies (3.5), then*

$$v(P_\mu^p) = v(P) \tag{3.6}$$

for $p > q$, where $q = \ln(m)/\ln[\min_{1\leq i\leq m}(b_i+1)/b_i]$.

A point to be emphasized is that the p-norm surrogate constraint method does not require a search for an optimal μ vector. The value of the μ vector can be simply assigned by solving (3.5).

Now we come back to Example 5.1 which the conventional surrogate constraint method fails to solve as we discussed before. Applying the p-norm surrogate constraint method yields the following formulation,

$$\begin{aligned} &\min\ 3x_1^2 + 2x_2^2 \\ &\text{s.t.}\ \ \{\mu_1^p[10-5x_1-2x_2]^p + \mu_2^p[15-2x_1-5x_2]^p\}^{1/p} \\ &\qquad \leq \{\mu_1^p \times 7^p + \mu_2^p \times 12^p\}^{1/p} \\ &\qquad x \in X. \end{aligned}$$

One normalized solution for $\mu_1 \times 7 = \mu_2 \times 12$ is $(\hat{\mu}_1, \hat{\mu}_2) = (0.6316, 0.3684)$. The value of B is equal to $\hat{\mu}_1 \times 7 = \hat{\mu}_2 \times 12 = 4.4211$. Figure 3.3 shows $S^p(\hat{\mu})$ for $p = 1, 2, 6, 9$. It can be clearly observed that when $p = 9$, $S^p(\hat{\mu}) = S$ and the p-norm surrogate method successfully identifies the optimal solution $x^* = (0, 2)^T$ with zero duality gap.

While the p-norm surrogate constraint method greatly simplifies the dual search at the upper level, i.e., there is no need to search for the optimal multiplier vector, the resulting surrogate relaxation problem at the lower level, in general, becomes intractable, when comparing with the conventional surrogate constraint

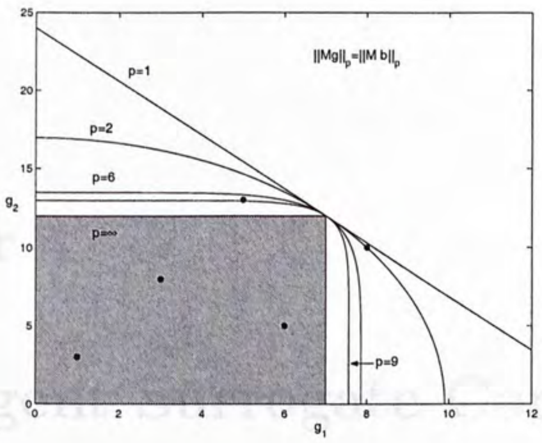


Figure 3.3: p-norm surrogate constraints in the constraint space of Example 5.1 with $\mu = (0.6316, 0.3684)^T$.

method. Concerning our separable primal problem (P), the p-norm surrogate constraint formulation (P_μ^p) is highly nonseparable which dynamic programming is not applicable to solve.

The surrogate dual was first investigated in [3] and [11] for continuous optimization problems. The surrogate dual was then applied to linear integer programming in [2][4][5]. Several surrogate dual search methods were developed for linear integer programming in [1][4][5][12].

The development of nonlinear surrogate constraint methods started with the p-norm surrogate method presented in [6]. Nonlinear surrogate constraint methods were also discussed in [7] and [10].

Chapter 4

Convergent Surrogate Constraint Dynamic Programming: Objective Level Cut

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 (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 (P_μ), the optimal solution of (P_μ) cannot be feasible. The solution concept presented in this chapter is to remove such an infeasible point which attains the optimality of (P_μ) from further consideration.

We require the integrality of f in this chapter in order to efficiently implement dynamic programming.

Assumption 4.1 *Function $f(x)$ is integer-valued, for all $x \in X$.*

Let us now consider the following surrogate constraint formulation with a lower bound constraint for the objective function,

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

where $c \in \mathbb{R}$. We assume that the value of μ is fixed. Note that problem $(P_\mu(c))$ is equivalent to the conventional surrogate constraint formulation (P_μ) when $c \le v(P_\mu)$. Let the feasible region of $(P_\mu(c))$ be $S(\mu; c)$.

The following theorem provides the basis for development of the convergent surrogate constraint dynamic programming using the concept of objective cut.

Theorem 4.1 *(i) When $v(P_\mu) \le c \le v(P)$,*

$$S \subseteq S(\mu; c) \subseteq S(\mu).$$

(ii) Let

$$\delta = \min\{v(P) - f(x) \mid x \in X \text{ and } f(x) < v(P)\}.$$

Problem $(P_\mu(c))$ is equivalent to the primal problem (P) when $v(P) - \delta < c \le v(P)$.

Proof. (i) The following is clear: If c is larger than $v(P_\mu)$, some points in $S(\mu) = \{x \in X \mid \mu^T(g(x)-b) \le 0\}$ could become infeasible in $S(\mu; c) = \{x \in X \mid \mu^T(g(x)-b) \le 0,\ f(x) \ge c\}$; If c is smaller than $v(P)$, some infeasible points of (P) could become feasible in $S(\mu; c)$.

(ii) When $v(P) - \delta < c \leq v(P)$, no infeasible point of (P) with its objective value less than $v(P)$ can be feasible in $S(\mu; c)$. At the same time all feasible points of (P) are still feasible in $S(\mu; c)$. □

Problem $(P_\mu(c))$ is a separable integer programming problem with 2 constraints which can be solved by dynamic programming in a relatively easy way compared to (P) when the number of constraints is large.

Note from (i) of Theorem 4.1, for any c satisfying $v(P_\mu) \leq c \leq v(P)$, $(P_\mu(c))$ is a relaxation of (P). Thus, the optimal value of $(P_\mu(c))$ is a lower bound of $v(P)$. The remaining task is how to adjust the value of c such that the iterative solution process of solving $(P_\mu(c))$ will eventually identify an optimal solution of (P).

In the proposed algorithm, the surrogate constraint problem (P_μ) is first solved for a selected μ. Dynamic programming is used to identify all the optimal solutions to (P_μ). If there is a feasible solution of (P) in the solution set of (P_μ), this solution is optimal to the primal problem (P). Otherwise, let $v_0 = v(P_\mu)$. By the weak duality, $v(P) > v_0$. Since $f(x)$ is integer-valued, $v(P)$ must be greater than or equal to $v_0 + 1$. Therefore, we form problem $(P_\mu(c))$ by incorporating the constraint of $f(x) \geq v_0 + 1$.

If there is a feasible solution of (P) in the solution set of $(P_\mu(v_0 + 1))$, this solution is optimal to the primal problem (P). Otherwise, let Let $v_1 = v(P_\mu(v_0 + 1))$. By the weak duality, $v(P) > v_1$. We solve then $(P_\mu(c))$ with $c = v_1 + 1$.

This process repeats and there must be a feasible solution of (P) that is optimal to $(P_\mu(c))$ when $c = v(P) - \delta + 1$.

We formally describe the algorithm as follows.

Procedure 4.1 *[Convergent Surrogate Constraint Dynamic Programming: Objective Level Cut]*

Step 0 Select a $\mu \in \mathbb{Z}_+^n$ and solve (P_μ). If there is a feasible solution of (P) in the solution set of (P_μ), this solution is optimal to the primal problem (P) and stop. Otherwise, let $v_0 = v(P_\mu)$ and $k = 1$.

Step 1 Using dynamic programming to solve the following problem $(P_\mu(v_{k-1} + 1))$,

$$\begin{aligned} \min \ & f(x) = \sum_{j=1}^{n} f_j(x_j) \\ \text{s.t.} \ & g_\mu(x) = \sum_{i=1}^{m} [\mu_i \sum_{i=1}^{n} g_{ij}(x_j)] \le \sum_{i=1}^{m} \mu_i b_i, \\ & \sum_{j=1}^{n} f_j(x_j) \ge v_{k-1} + 1, \\ & x \in X = X_1 \times X_2 \times \cdots \times X_n. \end{aligned}$$

Step 2 If there is a feasible solution of (P) in the solution set of $(P_\mu(v_{k-1} + 1))$, this solution is optimal to the primal problem (P) and stop. Otherwise, let $v_k = v(P_\mu(v_{k-1} + 1))$. Set $k = k + 1$ and go back to Step 1.

The solution concept of using objective level cut was first adopted in Lagrangian dual search [9] for nonlinear separable integer programming problems.

We illustrate the above solution procedure in the following example.

Example 4.1

$$\begin{aligned} \min \ & f(x) = 3x_1^2 + 2x_2^2 \\ \text{s.t.} \ & g_1(x) = -3x_1 + 2x_2 \le -4, \end{aligned}$$

$$\begin{aligned} g_2(x) &= x_1 + x_2 \leq 3, \\ g_3(x) &= 2x_1 + x_2 \leq 5, \\ g_4(x) &= x_1 - 2x_2 \leq 0, \\ g_5(x) &= -6x_1 + 5x_2 \leq -5, \\ x \in X &= \left\{x \in \mathbb{Z}^2 \mid 0 \leq x_i \leq 3,\ i = 1, 2\right\}. \end{aligned}$$

The optimal solution is $x^* = (2,1)^T$ with $f(x^*) = 14$.

The iteration process of Procedure 4.1 for this example is described as follows:

Step 0. Select $\mu_1 = \mu_2 = 1$, solve the following surrogate problem (P_μ) using dynamic programming,

$$\begin{aligned} \min\ & f(x) = 3x_1^2 + 2x_2^2 \\ \text{s.t.}\ & g_\mu(x) = -5x_1 + 7x_2 \leq -1, \\ & x \in X = \left\{x \in \mathbb{Z}^2 \mid 0 \leq x_i \leq 3,\ i = 1, 2\right\}. \end{aligned}$$

We obtain $x^0 = (1,0)^T$ and $f(x^0) = 3$. Solution x^0 is not feasible to the primal problem (P). Let $v_0 = f(x^0) = 3$ and $k = 1$.

Iteration 1

Step 1. Using dynamic programming to solve the problem $(P_\mu(v_0 + 1))$,

$$\begin{aligned} \min\ & f(x) = 3x_1^2 + 2x_2^2 \\ \text{s.t.}\ & g_\mu(x) = -5x_1 + 7x_2 \leq -1, \\ & -f(x) = -3x_1^2 - 2x_2^2 \leq -4, \\ & x \in X = \left\{x \in \mathbb{Z}^2 \mid 0 \leq x_i \leq 3,\ i = 1, 2\right\}, \end{aligned}$$

yields solution $x^1 = (2,0)^T$ with $f(x^1) = 12$.

Step 2. Solution x^1 is not feasible to the primal problem (P). Let $v_1 = f(x^1) = 12$ and $k = 2$.

Iteration 2

Step 1. Using dynamic programming to solve the problem $(P_\mu(v_1 + 1))$,

$$\begin{aligned} \min \ & f(x) = 3x_1^2 + 2x_2^2 \\ \text{s.t.} \ & g(x) = -5x_1 + 7x_2 \leq -1, \\ & -f(x) = -3x_1^2 - 2x_2^2 \leq -13, \\ & x \in X = \left\{x \in \mathbb{Z}^2 \mid 0 \leq x_i \leq 3, \ i = 1, 2\right\}, \end{aligned}$$

generates solution $x^2 = (2, 1)^T$ with $f(x^2) = 14$.

Step 2. Solution x^2 is feasible to the problem (P). Stop and the optimal solution to the primal problem is $x^* = x^2 = (2, 1)^T$ with $f(x^*) = 14$.

Chapter 5

Convergent Surrogate Constraint Dynamic Programming: Domain Cut

As we did in the previous chapter, the central solution concept of the proposed convergent surrogate constraint dynamic programming is to gradually remove some "active" infeasible solutions that attain optimal positions in the surrogate constraint formulation. Instead of imposing a lower bound on the objective function as we proposed in the last chapter, we propose a domain cut approach to cut off sub-domains that contain "active" infeasible solutions from further consideration.

Note that in the development of the current chapter, we do not require the integrality of the objective function f.

Let α, $\beta \in \mathbb{Z}^n$, where $\mathbb{Z}^n$ denotes the set of integer points in $\mathbb{R}^n$. Denote by $[\alpha, \beta]$ the box (hyper-rectangle) formed by α and β, $[\alpha, \beta] = \{x \mid \alpha_j \leq x_j \leq$

$\beta_j, j = 1, \ldots, n\}$. Let $\langle\alpha, \beta\rangle$ denote the set of integer points in $[\alpha, \beta]$,

$$\langle\alpha, \beta\rangle = \Pi_{j=1}^{n}\langle\alpha_j, \beta_j\rangle = \langle\alpha_1, \beta_1\rangle \times \langle\alpha_2, \beta_2\rangle \cdots \times \langle\alpha_n, \beta_n\rangle.$$

The set $\langle\alpha, \beta\rangle$ is called an *integer box.* For convenience, we define $[\alpha, \beta] = \langle\alpha, \beta\rangle = \emptyset$ if $\alpha \not\leq \beta$.

Let $\lfloor t \rfloor$ denote the maximum integer less than or equal to t and $\lceil t \rceil$ the minimum integer greater than or equal to t.

To illustrate the solution concept behind the proposed convergent surrogate constraint dynamic programming using domain cut, let us consider the following example.

Example 5.1

$$\begin{aligned}
\min\ & f(x) = 3x_1^2 + 2x_2^2 \\
\text{s.t.}\ & g_1(x) = 2x_1 + 3x_2 \leq 7, \\
& g_2(x) = (7 - 2x_1) + (8 - 2x_2) \leq 10, \\
& x \in X = \left\{x \in \mathbb{Z}^2 \mid 0 \leq x_i \leq 3,\ i = 1, 2\right\}.
\end{aligned}$$

Figures 5.1 and 5.2 illustrate the feasible regions in both the x-space and the g-space. Note there are only two feasible points in this example, $(3, 0)^T$ and $(2, 1)^T$.

Applying the conventional surrogate constraint method to solve the above example problem yields,

$$\begin{aligned}
\min\ & f(x) = 3x_1^2 + 2x_2^2 \\
\text{s.t.}\ & g_\mu(x) = \mu_1(2x_1 + 3x_2) + \mu_2(15 - 2x_1 - 2x_2) \leq 7\mu_1 + 10\mu_2, \\
& x \in X = \left\{x \in \mathbb{Z}^2 \mid 0 \leq x_i \leq 3,\ i = 1, 2\right\}.
\end{aligned}$$

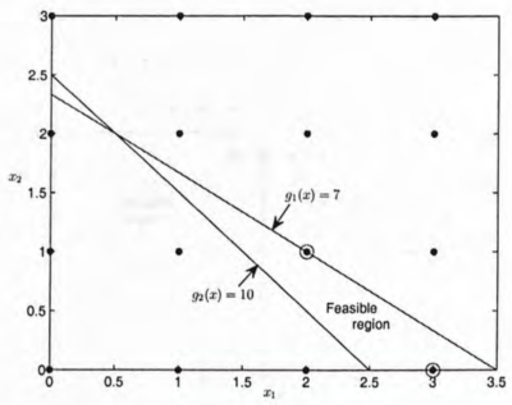


Figure 5.1: Feasible region in the x-space

Setting $\mu^* = (0,1)^T$ gives a feasible half-space in x and yields a solution $x^0 = (1,2)^T$ with $f(x^0) = 11$. See Figure 5.3. Solution $x^0 = (1,2)^T$ is infeasible with $g_1(x^0) = 8$ and $g_2(x^0) = 9$. Note that x^0 violates $g_1(x) \le 7$ and $g_1(x)$ is an increasing linear function of both x_1 and x_2. Thus, integer box $\langle (1,2)^T, (3,3)^T \rangle$ does not contain any feasible point and can be removed from X for further consideration. Let

$$X^1 = X^0 \setminus \langle (1,2)^T, (3,3)^T \rangle = X_1^1 \cup X_2^1 = \langle (0,0)^T, (0,3)^T \rangle \cup \langle (1,0)^T, (3,1)^T \rangle.$$

See Figure 5.4. Solving (P_{μ^*}) with $\mu^* = (0,1)^T$ on X_1^1 and X_2^1, respectively, yields a solution on X_1^1, $x_1^1 = (0,3)^T$, and a solution on X_2^1, $x_2^1 = (2,1)^T$. Note that $(2,1)^T$ is feasible with $f(x_2^1) = 14$. Set $(2,1)^T$ as the incumbent and remove X_2^1 from further consideration. Solution $(0,3)^T$ violates the first constraint and we cut $\langle (0,3)^T, (0,3)^T \rangle$ from X_1^1, resulting

$$X_1^2 = X_1^1 \setminus \langle (0,3)^T, (0,3)^T \rangle = \langle (0,0)^T, (0,2)^T \rangle$$

Problem (P_{μ^*}) with $\mu^* = (0,1)^T$ is infeasible on X_1^2 and X_1^2 is removed from

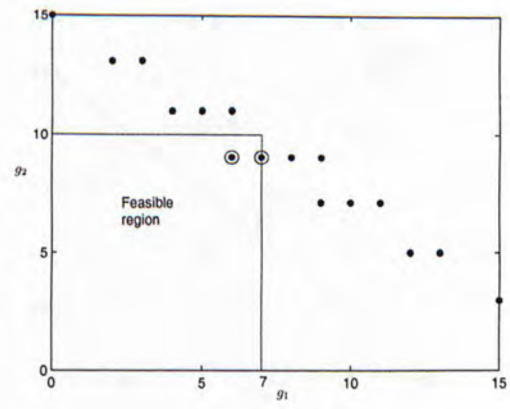


Figure 5.2: Feasible region in the g-space

further consideration. No more integer box is left and the solution process for the example terminates with the incumbent $(2,1)^T$ as the optimal solution with $v(P) = 14$.

The above example demonstrates that, when the duality gap is nonzero, domain cut can be implemented to cut off some infeasible integer subbox whose objective value dominates the feasible region in the surrogate constraint formulation. Carrying out such a procedure repeatedly gradually reduces the duality gap and eventually eliminates the duality. The solution of this iterative process converges to the solution of the original problem.

The following theorem gives conditions under which the above domain cut procedure can be applied.

Theorem 5.1 *Let $\tilde{x}$ be a solution to (P_μ) on $\langle\alpha,\beta\rangle$. If $\tilde{x}$ is infeasible to (P), more specifically, $g_i(\tilde{x}) > b_i$ for some $i \in \{1,\ldots,m\}$, then the following hold.*

(i) If f is concave, then the following integer subbox $\langle\gamma,\delta\rangle$ contains no feasible solution of (P), where for i = 1, ..., n, $\gamma_i = \tilde{x}_i$ and $\delta_i = \beta_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} < 0$, γ_i

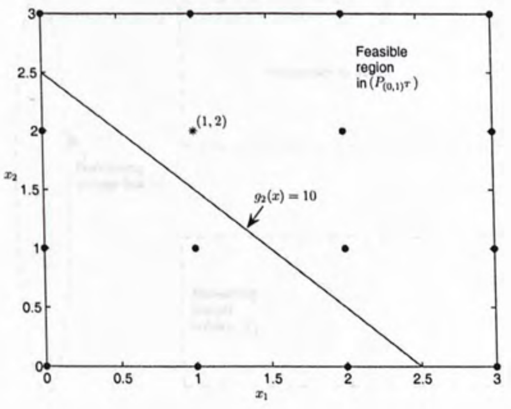


Figure 5.3: Feasible half-space resulted from the surrogate constraint method with $\mu^* = (0,1)^T$

$= \alpha_i$ and $\delta_i = \tilde{x}_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} > 0$, and $\gamma_i = \alpha_i$ and $\delta_i = \beta_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} = 0$. Thus, $\langle\gamma,\delta\rangle$ can be removed from $\langle\alpha,\beta\rangle$. See Figure 5.5.

(ii) *If f is a convex quadratic function taking the following form:*

$$f(x) = \sum_{j=1}^{n}(\frac{1}{2}c_j x_j^2 + d_j x_j)$$

with all $c_j > 0$, $j = 1, \ldots, n$, then the following integer subbox $\langle\gamma,\delta\rangle$, with

$$\begin{aligned} \gamma &= (\lceil -\frac{d_1}{c_1} - |\tilde{x}_1 + \frac{d_1}{c_1}|\rceil, \ldots, \lceil -\frac{d_n}{c_n} - |\tilde{x}_n + \frac{d_n}{c_n}|\rceil)^T, \quad (5.1) \\ \delta &= (\lfloor -\frac{d_1}{c_1} + |\tilde{x}_1 + \frac{d_1}{c_1}|\rfloor, \ldots, \lfloor -\frac{d_n}{c_n} + |\tilde{x}_n + \frac{d_n}{c_n}|\rfloor)^T, \quad (5.2) \end{aligned}$$

contains no feasible solution of (P) and can be removed from $\langle\alpha,\beta\rangle$. See Figure 5.6.

(iii) *If f is a concave quadratic function taking the following form:*

$$f(x) = \sum_{j=1}^{n}(\frac{1}{2}c_j x_j^2 + d_j x_j)$$

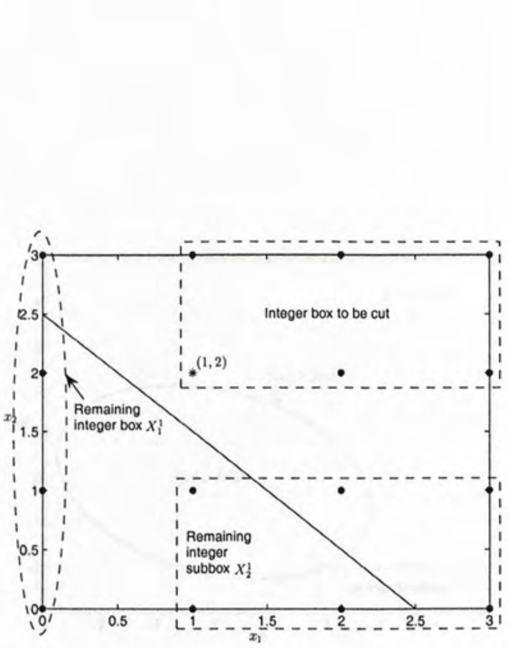


Figure 5.4: Domain cut in the 1st iteration

with all $c_j < 0$, $j = 1, \ldots, n$, then the optimal solution $\langle \alpha, \beta \rangle$ can only be in the following integer region $\langle \rho, \sigma \rangle \setminus \langle \gamma, \delta \rangle$ with

$$\begin{aligned} \rho &= (\lceil -d_1/c_1 - \sqrt{\left|2(f(\tilde{x}) + \sum_{j=1}^{n} d_j^2/(2c_j))/c_1\right|}\rceil, \ldots, \\ &\quad \lceil -d_n/c_n - \sqrt{\left|2(f(\tilde{x}) + \sum_{j=1}^{n} d_j^2/(2c_j))/c_n\right|}\rceil)^T, \\ \sigma &= (\lfloor -d_1/c_1 + \sqrt{\left|2(f(\tilde{x}) + \sum_{j=1}^{n} d_j^2/(2c_j))/c_1\right|}\rfloor, \ldots, \\ &\quad \lfloor -d_n/c_n + \sqrt{\left|2(f(\tilde{x}) + \sum_{j=1}^{n} d_j^2/(2c_j))/c_n\right|}\rfloor)^T, \end{aligned}$$

and, for $i = 1, \ldots, n$, $\gamma_i = \tilde{x}_i$ and $\delta_i = \beta_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} < 0$, $\gamma_i = \alpha_i$ and $\delta_i = \tilde{x}_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} > 0$, and $\gamma_i = \alpha_i$ and $\delta_i = \beta_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} = 0$. See Figure 5.7.

(iv) *If f is a monotone function of x, i.e., for all i, $i = 1, \ldots, n$, f is either increasing or decreasing with respect to x_i, the following integer subbox $\langle \gamma, \delta \rangle$ contains no feasible solution of (P), where for $i = 1, \ldots, n$, $\gamma_i = \tilde{x}_i$ and δ_i*

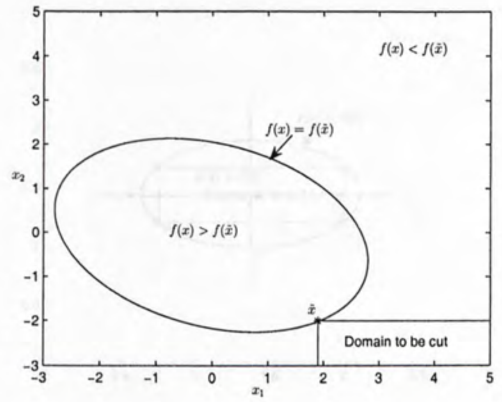


Figure 5.5: Domain cut when f is concave

$= \beta_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} < 0$, $\gamma_i = \alpha_i$ and $\delta_i = \tilde{x}_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} > 0$, and $\gamma_i = \alpha_i$ and $\delta_i = \beta_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} = 0$. Thus, $\langle \gamma, \delta \rangle$ can be removed from $\langle \alpha, \beta \rangle$. See Figure 5.8.

(v) *If g_i is convex, then the following integer subbox $\langle \gamma, \delta \rangle$ contains no feasible solution of (P), where for $i = 1, \ldots, n$, $\gamma_i = \tilde{x}_i$ and $\delta_i = \beta_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} > 0$, $\gamma_i = \alpha_i$ and $\delta_i = \tilde{x}_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} < 0$, and $\gamma_i = \alpha_i$ and $\delta_i = \beta_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} = 0$. Thus, $\langle \gamma, \delta \rangle$ can be removed from $\langle \alpha, \beta \rangle$. See Figure 5.9.*

(vi) *If g_i is a convex quadratic function of x taking the following form:*

$$g_i(x) = \sum_{j=1}^{n}(\frac{1}{2}c_{ij}x_j^2 + d_{ij}x_j)$$

with all $c_{ij} > 0$, $j = 1, \ldots, n$, then the optimal solution $\langle \alpha, \beta \rangle$ can only be in the following integer region $\langle \rho, \sigma \rangle \setminus \langle \gamma, \delta \rangle$ with

$$\begin{aligned} \rho &= (\lceil -d_{i1}/c_{i1} - \sqrt{\left|2(g_i(\tilde{x}) + \sum_{j=1}^{n} d_{ij}^2/(2c_{ij}))/c_{i1}\right|}\rceil, \ldots, \\ &\quad \lceil -d_{in}/c_{in} - \sqrt{\left|2(g_i(\tilde{x}) + \sum_{j=1}^{n} d_{ij}^2/(2c_{ij}))/c_{in}\right|}\rceil)^T, \end{aligned}$$

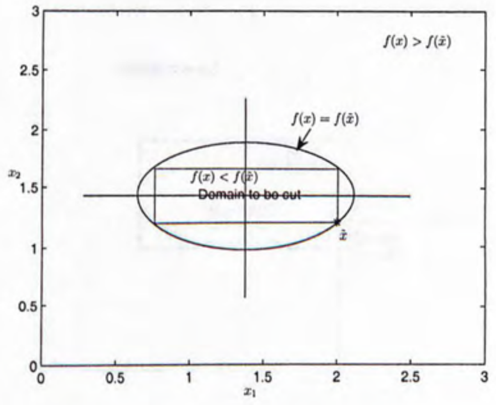


Figure 5.6: Domain cut when f is convex and quadratic

$$\begin{aligned}\sigma &= (\lfloor -d_{i1}/c_{i1} + \sqrt{\left|2(g_i(\tilde{x}) + \sum_{j=1}^{n} d_{ij}^2/(2c_{ij}))/c_{i1}\right|}\rfloor, \ldots, \\ &\quad \lfloor -d_{in}/c_{in} + \sqrt{\left|2(g_i(\tilde{x}) + \sum_{j=1}^{n} d_{ij}^2/(2c_{ij}))/c_{in}\right|}\rfloor)^T,\end{aligned}$$

and, for i = 1, ..., n, $\gamma_i = \tilde{x}_i$ *and* $\delta_i = \beta_i$ *if* $\frac{\partial g_i(\tilde{x})}{\partial x_i} > 0$, $\gamma_i = \alpha_i$ *and* $\delta_i = \tilde{x}_i$ *if* $\frac{\partial g_i(\tilde{x})}{\partial x_i} < 0$, *and* $\gamma_i = \alpha_i$ *and* $\delta_i = \beta_i$ *if* $\frac{\partial g_i(\tilde{x})}{\partial x_i} = 0$. *See Figure 5.10.*

(vii) *If* g_i *is a concave quadratic function of* x *taking the following form:*

$$g_i(x) = \sum_{j=1}^{n}(\frac{1}{2}c_{ij}x_j^2 + d_{ij}x_j)$$

with all $c_{ij} < 0$, $j = 1, \ldots, n$, *then the following integer subbox* $\langle\gamma, \delta\rangle$ *with*

$$\gamma = (\lceil -\frac{d_{i1}}{c_{i1}} - |\tilde{x}_1 + \frac{d_{i1}}{c_{i1}}|\rceil, \ldots, \lceil -\frac{d_{in}}{c_{in}} - |\tilde{x}_n + \frac{d_{in}}{c_{in}}|\rceil)^T, \quad (5.3)$$

$$\delta = (\lfloor -\frac{d_{i1}}{c_{i1}} + |\tilde{x}_1 + \frac{d_{i1}}{c_{i1}}|\rfloor, \ldots, \lfloor -\frac{d_{in}}{c_{in}} + |\tilde{x}_n + \frac{d_{in}}{c_{in}}|\rfloor)^T, \quad (5.4)$$

contains no feasible solution of (P). See Figure 5.11.

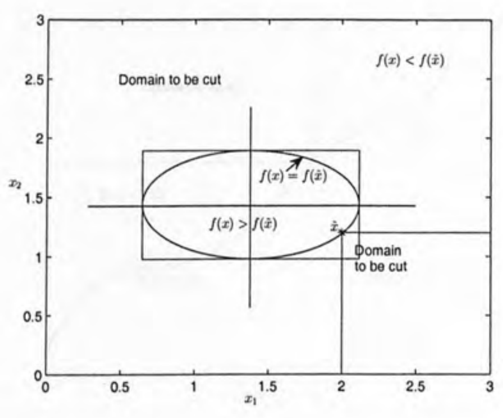


Figure 5.7: Domain cut when f is concave and quadratic

(viii) *If g_i is a monotone function of x, then the following integer subbox $\langle\gamma,\delta\rangle$ contains no feasible solution of (P), where for i = 1, ..., n, $\gamma_i = \tilde{x}_i$ and $\delta_i = \beta_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} > 0$, $\gamma_i = \alpha_i$ and $\delta_i = \tilde{x}_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} < 0$, and $\gamma_i = \alpha_i$ and $\delta_i = \beta_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} = 0$. Thus, $\langle\gamma,\delta\rangle$ can be removed from $\langle\alpha,\beta\rangle$. See Figure 5.12.*

Proof. We will give separate proofs for all of the above eight cases. The proofs are based on results from [8] and [9].

(i) When f is concave, the set $\{x \in \langle\alpha,\beta\rangle \mid f(x) \geq f(\tilde{x})\}$ is a convex set as shown in Figure 5.5, outside of which all points have an objective value strictly less than $f(\tilde{x})$. Note that $f(\tilde{x})$ is a lower bound of $v(P)$ on $\langle\alpha,\beta\rangle$ and, by the weaker duality, no point outside of $\{x \in \langle\alpha,\beta\rangle \mid f(x) \geq f(\tilde{x})\}$ can be optimal. Based on the sign of the normal vector of f at $\tilde{x}$, the box $\langle\gamma,\delta\rangle$, with, for i = 1, ..., n, $\gamma_i = \tilde{x}_i$ and $\delta_i = \beta_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} < 0$, $\gamma_i = \alpha_i$ and $\delta_i = \tilde{x}_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} > 0$, and $\gamma_i = \alpha_i$ and $\delta_i = \beta_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} = 0$, is outside of $\{x \in \langle\alpha,\beta\rangle \mid f(x) \geq f(\tilde{x})\}$ and can be removed.

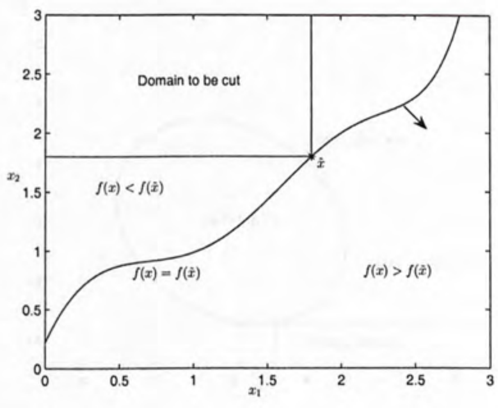


Figure 5.8: Domain cut when f is monotone

(ii) Consider the following ellipse contour of f:

$$f(x) = \sum_{j=1}^{n} [(1/2)c_j x_j^2 + d_j x_j] = f(\tilde{x}). \tag{5.5}$$

Clearly, the center of ellipse (5.5) is

$$o = (-d_1/c_1, \ldots, -d_n/c_n)^T. \tag{5.6}$$

See Figure 5.6. Let $E(\tilde{x})$ be the ellipsoid formed by the above ellipse contour. Since f is convex, all points inside $E(\tilde{x})$ possess an objective value smaller than $f(\tilde{x})$. By the weaker duality, no point inside $E(\tilde{x})$ can be feasible. By the symmetry of $E(\tilde{x})$, the integer box $\langle \gamma, \delta \rangle$, with

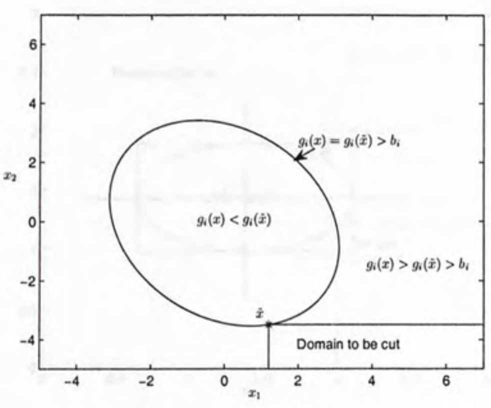
$$\gamma = (\lceil o_1 - |\tilde{x}_1 - o_1| \rceil, \ldots, \lceil o_n - |\tilde{x}_n - o_n| \rceil)^T, \tag{5.7}$$

$$\delta = (\lfloor o_1 + |\tilde{x}_1 - o_1| \rfloor, \ldots, \lfloor o_n + |\tilde{x}_n - o_n| \rfloor)^T, \tag{5.8}$$

is inside $E(\tilde{x})$ and can be removed.

(iii) Consider the following ellipse contour of f:

$$f(x) = \sum_{j=1}^{n} [(1/2)c_j x_j^2 + d_j x_j] = f(\tilde{x}). \tag{5.9}$$



$g_i(x) = g_i(\tilde{x}) > b_i$
$g_i(x) < g_i(\tilde{x})$
$g_i(x) > g_i(\tilde{x}) > b_i$
$\tilde{x}$
Domain to be cut
x_1
x_2

Figure 5.9: Domain cut when a convex g_i is violated

Clearly, the center of ellipse (5.9) is

$$o = (-d_1/c_1, \ldots, -d_n/c_n)^T \tag{5.10}$$

and the length of the i-th axis of ellipse (5.9) is

$$2r_i = 2\sqrt{\left|2(f(\tilde{x}) + \sum_{j=1}^{n} d_j^2/(2c_j))/c_i\right|}. \tag{5.11}$$

See Figure 5.7. Let $E(\tilde{x})$ be the ellipsoid formed by the above ellipse contour. Since f is concave, all points outside $E(\tilde{x})$ possess an objective value smaller than $f(\tilde{x})$. The minimum rectangle that encloses the ellipsoid $E(\tilde{x})$ is $[\rho, \sigma]$ with

$$\begin{aligned} \rho &= (o_1 - r_1, \ldots, o_n - r_n)^T, \\ \sigma &= (o_1 + r_1, \ldots, o_n + r_n)^T, \end{aligned}$$

and the optimal solution cannot be outside of this minimum rectangle. We can further cut off integer subbox $\langle \gamma, \delta \rangle$ from $\langle \rho, \sigma \rangle$ based on the argument given in Item (i).

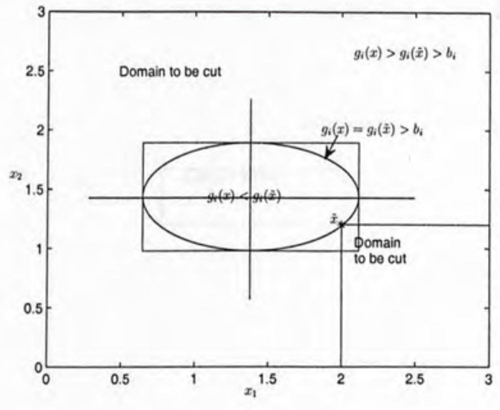

$g_i(x) > g_i(\tilde{x}) > b_i$
Domain to be cut
$g_i(x) = g_i(\tilde{x}) > b_i$
$g_i(x) < g_i(\tilde{x})$
$\tilde{x}$
Domain to be cut
x_1
x_2

Figure 5.10: Domain cut when g_i is convex and quadratic

(iv) If f is monotone, then any point in the subbox $\langle\gamma, \delta\rangle$ with, for $i = 1, \ldots, n$, $\gamma_i = \tilde{x}_i$ and $\delta_i = \beta_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} < 0$, $\gamma_i = \alpha_i$ and $\delta_i = \tilde{x}_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} > 0$, and $\gamma_i = \alpha_i$ and $\delta_i = \beta_i$ if $\frac{\partial f(\tilde{x})}{\partial x_i} = 0$, has an objective level not greater than $f(\tilde{x})$. Since $f(\tilde{x})$ is a lower bound of problem (P) on $\langle\alpha, \beta\rangle$, no point inside $\langle\gamma, \delta\rangle$ can be optimal. See Figure 5.8.

(v) When g_i is convex, the set $\{x \in \langle\alpha, \beta\rangle \mid g_i(x) \leq g_i(\tilde{x})\}$ is a convex set as shown in Figure 5.9, outside of which all points have a g_i value strictly larger than $g_i(\tilde{x})$. In other words, no point outside of $\{x \in \langle\alpha, \beta\rangle \mid g_i(x) \leq g_i(\tilde{x})\}$ can be feasible. Based on the sign of the normal vector of g_i at $\tilde{x}$, the box $\langle\gamma, \delta\rangle$, with, for $i = 1, \ldots, n$, $\gamma_i = \tilde{x}_i$ and $\delta_i = \beta_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} > 0$, $\gamma_i = \alpha_i$ and $\delta_i = \tilde{x}_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} < 0$, and $\gamma_i = \alpha_i$ and $\delta_i = \beta_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} = 0$, is outside of $\{x \in \langle\alpha, \beta\rangle \mid g_i(x) \leq g_i(\tilde{x})\}$ and can be removed.

(vi) Consider the following ellipse contour of g_i:

$$\sum_{j=1}^{n}[(1/2)c_{ij}x_j^2 + d_{ij}x_j] = g_i(\tilde{x}). \tag{5.12}$$

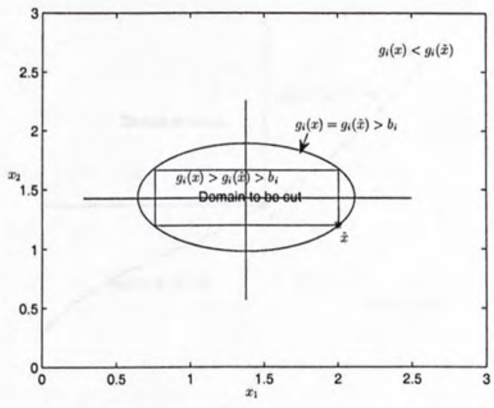

$g_i(x) < g_i(\tilde{x})$
$g_i(x) = g_i(\tilde{x}) > b_i$
$g_i(x) > g_i(\tilde{x}) > b_i$
Domain to be cut
$\tilde{x}$
x_1
x_2

Figure 5.11: Domain cut when g_i is concave and quadratic

Clearly, the center of ellipse (5.12) is

$$o = (-d_{i1}/c_{i1}, \ldots, -d_{in}/c_{in})^T \tag{5.13}$$

and the length of the j-th axis of ellipse (5.12) is

$$2r_j = 2\sqrt{\left|2(g_i(\tilde{x}) + \sum_{k=1}^{n} d_{ik}^2/(2c_{ik}))/c_{ij}\right|}. \tag{5.14}$$

See Figure 5.10. Let $E(\tilde{x})$ be the ellipsoid formed by the above ellipse contour. Since g_i is convex, all points outside $E(\tilde{x})$ possess a g_i value larger than $g_i(\tilde{x})$. The minimum rectangle that encloses the ellipsoid $E(\tilde{x})$ is $[\rho, \sigma]$ with

$$\begin{aligned} \rho &= (o_1 - r_1, \ldots, o_n - r_n)^T, \\ \sigma &= (o_1 + r_1, \ldots, o_n + r_n)^T, \end{aligned}$$

and the optimal solution cannot be outside of this minimum rectangle. We can further cut off integer subbox $\langle \gamma, \delta \rangle$ from $\langle \rho, \sigma \rangle$ based on the argument given in Item (v).

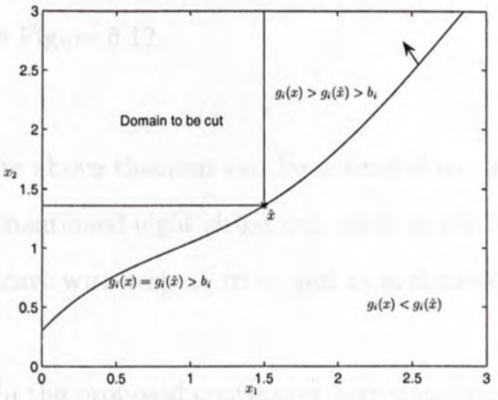

Domain to be cut
$g_i(x) > g_i(\tilde{x}) > b_i$
$\tilde{x}$
$g_i(x) = g_i(\tilde{x}) > b_i$
$g_i(x) < g_i(\tilde{x})$
x_1
x_2

Figure 5.12: Domain cut when g_i is monotone

(vii) Consider the following ellipse contour of g_i:

$$\sum_{j=1}^{n}[(1/2)c_{ij}x_j^2 + d_{ij}x_j] = g_i(\tilde{x}). \tag{5.15}$$

Clearly, the center of ellipse (5.15) is

$$o = (-d_{i1}/c_{i1}, \ldots, -d_{in}/c_{in})^T. \tag{5.16}$$

See Figure 5.11. Let $E(\tilde{x})$ be the ellipsoid formed by the above ellipse contour. Since g_i is concave, all points inside $E(\tilde{x})$ possess a g_i value larger than $g_i(\tilde{x})$, thus all infeasible. By the symmetry of $E(v\tilde{x})$, the integer box $\langle\gamma, \delta\rangle$, with

$$\gamma = (\lceil o_{i1} - |\tilde{x}_1 - o_{i1}|\rceil, \ldots, \lceil o_{in} - |\tilde{x}_n - o_{in}|\rceil)^T, \tag{5.17}$$

$$\delta = (\lfloor o_{i1} + |\tilde{x}_1 - o_{i1}|\rfloor, \ldots, \lfloor o_{in} + |\tilde{x}_n - o_{in}|\rfloor)^T, \tag{5.18}$$

is inside $E(\tilde{x})$ and can be removed.

(viii) If g_i is monotone, then any point in the subbox $\langle\gamma, \delta\rangle$ with, for $i = 1, \ldots, n$, $\gamma_i = \tilde{x}_i$ and $\delta_i = \beta_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} > 0$, $\gamma_i = \alpha_i$ and $\delta_i = \tilde{x}_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} < 0$, and

$\gamma_i = \alpha_i$ and $\delta_i = \beta_i$ if $\frac{\partial g_i(\tilde{x})}{\partial x_i} = 0$, has a g_i level not less than $g_i(\tilde{x})$, thus infeasible. See Figure 5.12.

□

Note that the above theorem can be extended to situations that do not fall into the above mentioned eight situations, such as a 4-dimensional situation where f is only concave with respect to x_1 and x_2 and monotone with respect to x_3 and x_4.

A key issue in the proposed convergent surrogate constraint dynamic programming method is how to partition a non-rectangular domain into a union of integer boxes such that the surrogate constraint dynamic programming can be applied to every newly generated integer subbox after a cutting process. We have the following result.

Lemma 5.1 *[9] Let $A = \langle \alpha, \beta \rangle$ and $B = \langle \gamma, \delta \rangle$, where α, β, γ, $\delta \in \mathbb{Z}^n$ and $\alpha \le \gamma \le \delta \le \beta$. Then $A \setminus B$ can be partitioned into at most $2n$ integer boxes.*

$$\begin{aligned} A \setminus B &= \left\{ \cup_{j=1}^{n} \left(\Pi_{i=1}^{j-1} \langle \alpha_i, \delta_i \rangle \times \langle \delta_j + 1, \beta_j \rangle \times \Pi_{i=j+1}^{n} \langle \alpha_i, \beta_i \rangle \right) \right\} \qquad (5.19) \\ &\quad \cup \left\{ \cup_{j=1}^{n} \left(\Pi_{i=1}^{j-1} \langle \gamma_i, \delta_i \rangle \times \langle \alpha_j, \gamma_j - 1 \rangle \times \Pi_{i=j+1}^{n} \langle \alpha_i, \delta_i \rangle \right) \right\}. \end{aligned}$$

Now we formally describe the algorithm.

Procedure 5.1 *[Convergent Surrogate Constraint Dynamic Programming: Domain Cut]*

Step 0 (Initialization). Select a surrogate multiplier μ and use dynamic programming to solve (P_μ). Let x^0 be the solution to (P_μ). If x^0 is feasible, then x^0 is the optimal solution, stop. Otherwise, calculate $f(x^0)$. Let $X^0 = X$, $k = 0$, and $f_{opt} = -\infty$.

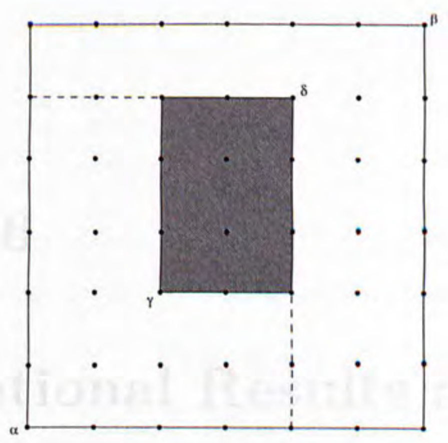


Figure 5.13: Partition of $A \setminus B$.

Step 1 (Sub-Domain Selection) Select an integer subbox X^{kj} from X^k with the smallest objective value of (P_μ), $f(x^{kj})$. Let $X^k = X^k \setminus X^{kj}$.

Step 2 (Cut and Partition) Cut out from X^{kj} certain integer boxes of infeasible solutions that include x^{kj} using one of the formulae in Theorem 5.1 and partition the remaining domain, Z^k, into a union of integer sub-boxes.

Step 3 (Evaluation) Solve (P_μ) on every integer subbox in Z^k. Remove all the integer subboxes from Z^k whose solution is feasible in (P). Update x_{opt} and f_{opt} if a feasible solution found possesses an objective function value smaller than f_{opt}. Let $X^{k+1} = X^k \cup Z^k$.

Step 4 (Fathoming) Remove all the integer subboxes in X^{k+1} whose objective function value is larger than f_{opt}.

Step 5 (Optimality Check and Termination) If X^{k+1} is empty, stop and x_{opt} is optimal to (P) with f_{opt} as the objective function value. Otherwise, set $k = k + 1$, go back to Step 1.

Chapter 6

Computational Results and Analysis

We have proposed in the previous chapters two convergent surrogate constraint dynamic programming algorithms using objective level cut and domain cut, respectively, and we are now in a position to check whether they work or not for various testing problems and how each of them helps to mitigate the "curse of dimensionality". This chapter serves for the purpose of reporting computational results. Several testing problems are constructed and they are solved by the conventional dynamic programming and two convergent surrogate constraint dynamic programming algorithms proposed in this thesis. Various useful data are collected in the computational process. Specially, the statistics on the total number of state iterations for each problem while applying the three algorithms are analyzed. The ranges of states s_k, $k = 1, ..., n-1$ are calculated using (2.2) and (2.3). During the solution process, each s_k iterates from $\underline{s}_k$ to $\bar{s}_k$. The total number of state iterations for a problem while applying a particular algorithm can be obtained by summing up the number of states s_k for all k each time dynamic programming is applied and then summing up further all the subtotals. As the computational and storage requirements are almost linear functions of the number of state iterations, the comparison of the total number of state iterations with respect to the three algorithms will give us some insight on the effectiveness of the convergent surrogate constraint dynamic programming methods in mitigating the "curse of dimensionality". Computer codes are written in MATLAB and can be obtained from the CD-Rom submitted.

6.1 Sample problems

Below we will list some sample problems solved in our computational experiments. Note that we have set the surrogate multipliers μ_i, $i = 1, 2, ..., m$ all equal to 1 in the surrogate constraint formulation for all the test problems.

Example 6.1 *We consider Example 4.1 again.*

When adopting the conventional dynamic programming, we have $\underline{s}_2 = (-9, 0, 0, 0, -18)^T$, $\bar{s}_2 = (-4, 3, 5, 3, -5)^T$, $s_1 = (0, 0, 0, 0, 0)^T$. Thus, we need totally $6 \times 4 \times 6 \times 4 \times 14 + 1 = 8065$ state iterations.

When adopting the convergent surrogate constraint dynamic programming with objective level cut, we apply dynamic programming once to solve the surrogate constraint problem and then apply dynamic programming twice to solve successively two doubly-constrained problems $(P_\mu(v_0 + 1))$ and $(P_\mu(v_1 + 1))$ with an objective level cut. For the surrogate constraint problem, $\underline{s}_2 = -15$, $\bar{s}_2 = -1$, $s_1 = 0$. For the two problems with an objective level cut, $\underline{s}_2 = (-15, -27)^T$, $\bar{s}_2 = (-1, 0)^T$ and $s_1 = (0, 0)^T$. Thus, each has $15 \times 28 + 1 = 421$ state iterations. There are $(16 + 421 \times 2) = 858$ state iterations in total.

When adopting the convergent surrogate constraint dynamic programming with domain cut, we apply dynamic programming 3 times to solve the singly constrained surrogate constrained problem on integer boxes $\langle(0,0)^T,(3,3)^T\rangle$, $\langle(2,0)^T,(3,3)^T\rangle$ and $\langle(2,1)^T,(3,3)^T\rangle$, respectively. For the first integer box, $\underline{s}_2=-15$, $\bar{s}_2=-1$. For the second or the third integer box, $\underline{s}_2=-15$, $\bar{s}_2=-10$. Thus, we, in total, require $(15+1)+(6+1)+(6+1)=30$ state iterations.

Example 6.2

$$\begin{aligned}
\min\ & f(x)=3x_1^2+2x_2^2+5x_3^2+x_4^2+4x_5^2\\
\text{s.t. }\ & g_1(x)=x_1^2+x_2^2+x_3^2+x_4^2+x_5^2\le 20,\\
& g_2(x)=x_1^2+x_2^2-10x_1+x_3^2+x_4^2+x_5^2-12x_5\le -20,\\
& g_3(x)=x_1^2-6x_1+2x_2^2+x_3^2+x_4^2-4x_4+x_5^2\le 0,\\
& x\in X=\left\{x\in\mathbb{Z}^5 \mid 0\le x_i\le 3,\ i=1,2,3,4,5\right\}.
\end{aligned}$$

The solution is $x^*=(1,2,0,2,1)^T$ and $f(x^*)=19$. With conventional dynamic programming, $\underline{s}_5=(0,-21,-13)^T$, $\bar{s}_5=(20,7,0)^T$, $\underline{s}_4=(0,-21,-9)^T$, $\bar{s}_4=(20,7,4)^T$, $\underline{s}_3=(0,-21,-9)^T$, $\bar{s}_3=(18,7,4)^T$, $\underline{s}_2=(0,0,-9)^T$, $\bar{s}_2=(9,9,0)^T$ and $s_1=(0,0,0)^T$. Therefore, we need in total $21\times 29\times 14+21\times 29\times 14+19\times 29\times 14+10\times 10\times 10+1=25,767$ state iterations.

The surrogate constraint formulation of this problem is

$$\begin{aligned}
\min\ & f(x)=3x_1^2+2x_2^2+5x_3^2+x_4^2+4x_5^2\\
\text{s.t. }\ & g(x)=3x_1^2-6x_1+4x_2^2-10x_2+3x_3^2+3x_4^2-4x_4+3x_5^2-12x_5\le 0,\\
& x\in X=X_1\times X_2\times X_3\times X_4\times X_5.
\end{aligned}$$

When adopting the convergent surrogate constraint dynamic programming with objective level cut, we apply dynamic programming once to the surrogate constraint problem and then to nineteen 2-constraint problems $(P_\mu(v_k+1))$, $k = 0, 1, ..., 18$. For the surrogate problem, $\underline{s}_5 = -10$, $\bar{s}_5 = 12$, $\underline{s}_4 = -9$, $\bar{s}_4 = 13$, $\underline{s}_3 = -9$, $\bar{s}_3 = 13$, $\underline{s}_2 = -3$, $\bar{s}_2 = 9$, $s_1 = 0$, thus leading to 23+23+23+13+1 = 83 state iterations. For each of the 2-constraint problems $(P_\mu(v_k+1))$, $\underline{s}_5 = (-10, -99)^T$, $\bar{s}_5 = (12, 0)^T$, $\underline{s}_4 = (-9, -90)^T$, $\bar{s}_4 = (13, 0)^T$, $\underline{s}_3 = (-9, -45)^T$, $\bar{s}_3 = (13, 0)^T$, $\underline{s}_2 = (-3, -27)^T$, $\bar{s}_2 = (9, 0)^T$, $s_1 = (0, 0)^T$, thus leading to $23 \times 100 + 23 \times 91 + 23 \times 46 + 13 \times 28 + 1 = 5,816$ state iterations each. There are totally $83 + 5816 \times 19 = 110,587$ state iterations.

When adopting the convergent surrogate constraint dynamic programming with domain cut, we apply dynamic programming to the singly constrained surrogate constraint problem on 23 integer boxes involved in the solution process. Among them, 10 requires state iterations between 0 and 50, and the other 13 requires state iterations between 50 and 100. The total number of state iterations is 1,263.

Example 6.3

$$
\begin{aligned}
\min\ & f(x) = 3x_1^2 + 2x_2^2 + 5x_3^2 \\
\text{s.t.}\ & g_1(x) = x_1 - x_2 + x_3 \le 2, \\
& g_2(x) = -2x_1 - x_2 - x_3 \le -8, \\
& g_3(x) = -5x_1 + 2x_2 + x_3 \le -1, \\
& g_4(x) = 3x_1 - 2x_2 - x_3 \le -3, \\
& g_5(x) = x_1^2 + x_2^2 - 6x_2 + x_3^2 - 4x_3 \le -8, \\
& g_6(x) = 4x_1^2 - 20x_1 + x_2^2 - 4x_2 + 4x_3^2 - 28x_3 \le -75, \\
& g_7(x) = 9x_1^2 - 33x_1 + x_2^2 \le -20, \\
& g_8(x) = x_1^2 - 2x_1 + x_3^2 - 7x_3 \le -10,
\end{aligned}
$$

$$x \in X = \left\{ x \in \mathbb{Z}^3 \mid 0 \le x_i \le 3,\ i = 1, 2, 3 \right\}.$$

The solution is $x^* = (2, 3, 3)^T$ and $f(x^*) = 75$. With conventional dynamic programming, $\underline{s}_3 = (-3, -9, -15, -6, -9, -28, -30, -1)^T$, $\bar{s}_3 = (2, -5, -1, 0, -4, -27, -20, 2)^T$, $\underline{s}_2 = (0, -6, -15, 0, 0, -24, -30, -1)^T$, $\bar{s}_2 = (3, -2, -1, 6, 5, -23, -20, 2)^T$ and $s_1 = (0, 0, 0, 0, 0, 0, 0, 0)^T$. We need 2,772,001 state iterations in total.

The surrogate constraint formulation of this problem is

$$\begin{aligned}
&\min\ f(x) = 3x_1^2 + 2x_2^2 + 5x_3^2 \\
&\text{s.t.}\ \ g(x) = 15x_1^2 - 58x_1 + 3x_2^2 - 12x_2 + 6x_3^2 - 39x_3 \le -123, \\
&\qquad x \in X = X_1 \times X_2 \times X_3.
\end{aligned}$$

When adopting the convergent surrogate constraint dynamic programming with objective level cut, we apply dynamic programming to the singly-constrained surrogate constraint problem first and two doubly constrained problems, $(P_\mu(v_k0 + 1))$ and $(P_\mu(v_1 + 1))$ successively. For the surrogate constraint problem, $\underline{s}_3 = -68$, $\bar{s}_3 = -60$, $\underline{s}_2 = -56$, $\bar{s}_2 = -48$, $s_1 = 0$, thus leading to 19 state iterations. For $(P_\mu(v_0 + 1))$, $\underline{s}_3 = (-68, -45)^T$, $\bar{s}_3 = (-60, -15)^T$, $\underline{s}_2 = (-56, -27)^T$, $\bar{s}_3 = (-48, 0)^T$, $s_1 = (0, 0)^T$, leading to 532 state iterations. For $(P_\mu(v_1 + 1))$, $\underline{s}_3 = (-68, -45)^T$, $\bar{s}_3 = (-60, -21)^T$, $\underline{s}_2 = (-56, -27)^T$, $\bar{s}_3 = (-48, -3)^T$, $s_1 = (0, 0)^T$, leading to 451 state iterations. Summing up, there are 1,002 state iterations.

When adopting the convergent surrogate constraint dynamic programming with domain cut, not taken into account the integer boxes being cut, there are 7 integer boxes involved in the solution process. Among them, 4 does not contain any feasible solution for the surrogate constraint problem and dynamic programming is applied to the remaining 3 integer boxes. We require 19, 19 and 13 state iterations on these three integer boxes, respectively. In total, 51 state iterations are required.

Example 6.4

$$\begin{aligned}
\min\ & f(x) = -3x_1^2 - 2x_2^2 - 5x_3^2 - 4x_4^2 \\
\text{s.t.}\ & g_1(x) = x_1 + x_2 + x_3 + x_4 \le 8, \\
& g_2(x) = x_1^2 + x_2 + x_3^2 + x_4^2 \le 12, \\
& g_3(x) = 3x_1 + x_2 - 2x_3 - x_4 \le 0, \\
& g_4(x) = x_1^2 - x_2 + 3x_3 + x_4 \le 6, \\
& g_5(x) = 5x_1^2 - 2x_2^2 + 3x_3^2 + x_4 \le 1, \\
& x \in X = \left\{x \in \mathbb{Z}^4 \mid 0 \le x_i \le 3,\ i = 1,2,3,4\right\}.
\end{aligned}$$

The solution is $x^* = (1,3,2,2)^T$ and $f(x^*) = -57$. With conventional dynamic programming, $\underline{s}_4 = (0,0,-6,-3,-18)^T$, $\bar{s}_4 = (8,12,3,6,1)^T$, $\underline{s}_3 = (0,0,0,-3,-18)^T$, $\bar{s}_3 = (6,12,9,6,1)^T$, $\underline{s}_2 = (0,0,0,0,0)^T$, $\bar{s}_2 = (3,9,9,9,19)^T$ and $s_1 = (0,0,0,0,0)^T$. There are totally 496,001 state iterations.

The surrogate constraint formulation of this problem is

$$\begin{aligned}
\min\ & f(x) = -3x_1^2 - 2x_2^2 - 5x_3^2 - 4x_4^2 \\
\text{s.t.}\ & g(x) = 7x_1^2 + 4x_1 - 2x_2^2 + 2x_2 + 4x_3^2 + 2x_3 + x_4^2 + 2x_4 \le 27, \\
& x \in X = X_1 \times X_2 \times X_3 \times X_4.
\end{aligned}$$

When adopting the convergent surrogate constraint dynamic programming with objective level cut, we apply dynamic programming first to the surrogate constraint problem and then to three two-constraint problems, $(P_\mu(v_0+1))$, $(P_\mu(v_1+1))$ and $(P_\mu(v_2+1))$ successively. For the surrogate constraint problem,

$\underline{s}_4 = -12$, $\bar{s}_4 = 27$, $\underline{s}_3 = -12$, $\bar{s}_3 = 27$, $\underline{s}_2 = 0$, $\bar{s}_3 = 39$, $s_1 = 0$, thus leading to 121 state iterations. For $(P_\mu(v_0 + 1))$, $\underline{s}_4 = (-12, 0)^T$, $\bar{s}_4 = (27, 73)^T$, $\underline{s}_3 = (-12, 0)^T$, $\bar{s}_3 = (27, 45)^T$, $\underline{s}_2 = (0, 0)^T$, $\bar{s}_3 = (39, 27)^T$, $s_1 = (0, 0)^T$, leading to 5921 state iterations. For $(P_\mu(v_1 + 1))$, $\underline{s}_4 = (-12, 0)^T$, $\bar{s}_4 = (27, 61)^T$, $\underline{s}_3 = (-12, 0)^T$, $\bar{s}_3 = (27, 45)^T$, $\underline{s}_2 = (0, 0)^T$, $\bar{s}_3 = (39, 27)^T$, $s_1 = (0, 0)^T$, leading to 5441 state iterations. For $(P_\mu(v_2 + 1))$, $\underline{s}_4 = (-12, 0)^T$, $\bar{s}_4 = (27, 58)^T$, $\underline{s}_3 = (-12, 0)^T$, $\bar{s}_3 = (27, 45)^T$, $\underline{s}_2 = (0, 0)^T$, $\bar{s}_3 = (39, 27)^T$, $s_1 = (0, 0)^T$, leading to 5321 state iterations. Summing up, there are in total 16,804 state iterations.

When adopting the convergent surrogate constraint dynamic programming with domain cut, not taken into account the integer boxes being cut, there are 10 integer boxes involved in the solution process. Dynamic programming is applied 10 times to the surrogate constraint problem with different feasible region. Among the 10 times, 3 requires less than 10 state iterations, 2 requires more than 100 state iterations, and 5 requires between 50 and 100 state iterations. In total, 664 state iterations are required.

From our more computational experiences, we conclude that with conventional dynamic programming, as the number of constraints, the number of variables or the range of variables increases, the amount of state iterations will increase correspondingly. Among them, the increment of the number of constraints inserts the major impact, interacting actively with the increment of the range of states, makes the number of state iterations soon become excessive with exponential increase, leading to the previously discussed "curse of dimensionality".

The following table compares the number of state iterations required in solving the above examples using the conventional dynamic programming and the convergent surrogate constraint dynamic programming with objective level cut and with domain cut, respectively.

Table 6.1: Comparison of the number of state iterations

	Conventional dynamic programming	Objective level cut	Domain cut
Example 4.1	8,065	858	30
Example 6.2	25,767	110,587	1,263
Example 6.3	2,772,001	1,002	51
Example 6.4	496,001	16,804	664

From Table 6.1, we can see that the convergent surrogate constraint dynamic programming with domain cut requires the least state iterations. Also, the convergent surrogate constraint dynamic programming with objective level cut generally requires fewer state iterations than the conventional dynamic programming. One obvious reason is that the objective level cut reduces the number of constraints to 2 in all subproblems following the surrogate constraint problem during the solution process. Similarly, for the convergent surrogate constraint dynamic programming with domain cut, the surrogate constraint formulation reduces the number of constraints to 1. To evaluate how the reduction in the number of constraints helps in mitigating the "curse of dimensionality", we observe from the calculations in the above examples that the number of state iterations each time dynamic programming is applied can be calculated as $\sum_{k=2}^{n}\Pi_{i=1}^{m}(\bar{s}_k(i)-\underline{s}_k(i)+1)+1$, where n is the number of variables, m is the number of constraints, and i is the index denoting the position in the m-dimensional array of the range of states. The reduction in the number of constraints m reduces the number of multiplying terms and thus alleviates the effect of exponential increase in the number of state iterations, mitigating the "curse of dimensionality".

However, in Example 6.2, the convergent surrogate constraint dynamic programming with objective level cut requires more state iterations than the conventional dynamic programming. This is because the convergence of objective level cut can be very slow. In the extreme case, each time the increment of objective value is only 1 for a sequence of problems $(P_\mu(v_k+1))$ before the optimal solution is obtained. The problems $(P_\mu(v_k+1))$ are two-constraint problems, and dynamic programming is applied to solve these problems. If the number of problems is large, the total number of state iterations may exceed what when applying conventional dynamic programming. On the other hand, the convergence of the convergent surrogate constraint dynamic programming with domain cut can also be slow. There may be lots of integer boxes generated in the solution process, and dynamic programming is applied on each of the boxes. However, the problems to be solved with respect to these integer boxes are one-constrained ones. Since dynamic programming is applied to solve a sequence of problems with only one constraint, even though the number of problems involved is large, the total number of state iterations is still acceptable. Therefore, the convergent surrogate constraint dynamic programming with domain cut generally performs better than the convergent surrogate constraint dynamic programming with objective level cut and the conventional dynamic programming.

Before we close this chapter, we examine another example to compare effectiveness of the two convergent surrogate constraint dynamic programming algorithms.

Example 6.5

$$\min\ f(x) = 2x_1^2 + 5x_2^2 + 4x_3^2 + 3x_4^2 + 6x_5^2 + 2x_6^2 + 4x_7^2 + 3x_8^2 + x_9^2 + 7x_{10}^2$$

$$\begin{aligned}
\text{s.t.}\quad & g_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2 + x_9^2 + x_{10}^2 \le 75, \\
& g_2(x) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \le 25, \\
& g_3(x) = -3x_1 + x_2 - 5x_3 + 2x_4 + x_5 + 3x_6 + 5x_7 - 2x_8 - 7x_9 + x_{10} \le -50, \\
& g_4(x) = x_1^2 - 6x_1 + x_2^2 + x_3^2 - 8x_3 + x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2 + x_9^2 - 6x_9 + x_{10}^2 \le -5, \\
& g_5(x) = 9x_3^2 - 42x_3 + x_9^2 - 6x_9 \le -50, \\
& x \in X = \left\{ x \in \mathbb{Z}^{10} \mid 0 \le x_i \le 5,\ i = 1, 2, ..., 10 \right\}.
\end{aligned}$$

The optimal solution of this example is $x^* = (1,0,2,0,0,0,0,1,5,0)^T$ and $f(x^*) = 46$. If conventional dynamic programming is applied to solve the problems, 110,037,271 state iterations are required to get the optimal solution. Otherwise, if the convergent surrogate constraint dynamic programming with objective cut is applied, the surrogate problem is solved first, which requires 764 state iterations. Then, the problem $(P_\mu(v_0 + 1))$ is solved, and it requires 376,539 state iterations. The first two iterations already require 377,303 state iterations, and the corresponding function value is still far from the optimal one, and more problems $(P_\mu(v_k + 1))$ are required to be solved before the optimal solution can be obtained. Compared with the conventional dynamic programming, the "curse of dimensionality" is only mitigated to a limited extent. However, solution can be obtained efficiently with the convergent surrogate constraint dynamic programming with domain cut. Not taken into account the integer boxes cut, there are 299 integer boxes involved in the solution process. Among them, 2 does not contain any feasible solution for the surrogate problem and dynamic programming is applied to the remaining 297 integer boxes. Among the 297 times dynamic programming is applied, 171 requires between 500 and 1000 state iterations, 112 requires between 100 and 500 state iterations, and 14 requires less than 100 state iterations. In total, 135542 state iterations are required.

Chapter 7

Conclusions

In this thesis, we have focused our efforts on devising efficient ways to mitigate the "curse of dimensionality". Two convergent surrogate constraint dynamic programming algorithms with objective level cut and domain cut have been proposed.

The motivation behind the novel convergent surrogate constraint dynamic programming algorithms is to alleviate the "curse of dimensionality" by working successively on singly or doubly constrained problems. The original curse of dimensionality exhibits when the number of constraints is high. However, dynamic programming, when it is applicatable, needs to be applied only once to obtain the solution. In the newly proposed approaches, we convert the curse of dimensionality in the state space to the number of subproblems to be solved in an iterative solution process. In the convergent surrogate constraint dynamic programming algorithm with objective level cut, the doubly constrained problem resulted from combining the surrogate constraint formulation with an objective level constraint has to be solved successively for a sequence of updated lower bound of the objective level, while in the convergent surrogate constraint dynamic programming algorithm with domain cut, the domain of decision variables is decomposed into sub-domains and dynamic programming needs to be applied on each sub-domain to a singly constrained (surrogate constraint) formulation. The migration of dimensionality in the above two schemes seems workable as evidenced from our numerical experiments.

In the proposed convergent surrogate constraint dynamic programming algorithm with objective level cut, one constraint is to enforce the objective function to exceed an increasing threshold, thus making the duality gap shrink. There are two research issues which are worth further efforts to investigate. First, we don't need to solve the dynamic programming problem starting from a scratch after the lower bound of the objective level is updated. We may devise an algorithm to use the information from the previous iterations in order to reduce the computational efforts. Second, it is even possible to design a singly-constrained dynamic programming algorithm to realize the solution for this special doubly constrained dynamic programming problem.

Bibliography

[1] M. E. Dyer, *Calculating surrogate constraints*, Mathematical Programming, vol 19, 1980, 255-278.

[2] B. Gavish, F. Glover, and H. Pirkul, *Surrogate constraints in integer programming*, Journal of Information and Optimization Sciences, vol 12, 1991, 219-228.

[3] H. J. Greenberg and W. P. Pierskalla, *Surrogate mathematical programming*, Operations Research, vol 18, 1970, 924-939.

[4] M. H. Karwan and R. L. Rardin, *Searchability of the composite and multiple surrogate dual functions*, Operations Research, vol 28, 1980, 1251-1257.

[5] M. H. Karwan and R. L. Rardin, *Surrogate dual multiplier search procedures in integer programming*, Operations Research, vol 32, 1984, 52-69.

[6] D. Li, *Zero duality gap in integer programming: p-norm Surrogate Constraint Method*, Operations Research Letters, vol 25, 1999, 89-96.

[7] D. Li and X. L. Sun, *Success guarantee of dual search in integer programming: p-th power Lagrangian method*, Journal of Global Optimization, vol 18, 2000, 235-254.

[8] D. Li, X. L. Sun, and F. L. Wang, *Convergent lagrangian and contour cut method for nonlinear integer programming with a quadratic objective function*, SIAM Journal on Optimization, vol 17, 2006, 372-400.

[9] D. Li and X. L. Sun, *Nonlinear Integer Programming*, Springer, 2006.

[10] D. Li and D. J. White, *p-th power lagrangian method for integer programming*, Annals of Operations Research, vol 98, 2000, 151-170.

[11] D. G. Luenberger. *Quasi-convex programming*, SIAM Journal on Applied Mathematics, vol. 16, 1968, 1090-1095.

[12] S. Sarin, M. H. Karwan, and R. L. Rardin, *A new surrogate dual multiplier search procedure*, Naval Research Logistics, vol 34, 1987, 431-450.

[13] D. Mayne, *A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems*, International Journal of Control, vol. 3, 1966, 85-95.

[14] R. E. Larson and A. J. Korsak, *A dynamic programming successive approximations technique with convergence proofs*, Automatica, vol 6, 1970, 245-252.

[15] T. L. Morin and A. M. O. Esogbue, *The imbedded state space approach to reducing dimensionality in dynamic programs of higher dimensions*, Journal of Mathematical Analysis and Applications, vol 48, 1974, 801-810.

[16] A. Haurie and P. L'Ecuyer, *Approxiation and bounds in discrete event dynamic programming*, IEEE Transactions on Automatic Control, vol 31, 1986, 227-235.

[17] L. Z. Liao and C. A. Shoemaker, *Convergence in unconstained discrete-time differential dynamic programming*, IEEE Transactions on Automatic Control, vol 36, 1991, 692-706.