Dynamic Programming using local optimality conditions for action elimination

Erich Wolfgang Steiner

Doctor of Philosophy
University of Edinburgh
1999

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Erich Wolfgang Steiner*)

Acknowledgements

Firstly, I would like to thank Dr. Ken McKinnon who has supervised my research throughout, and with whom it has been a pleasure to work, both on a personal level and work-wise. With his thoroughness and different ways of looking at mathematical issues he has provided me with a lot of motivation.

Thanks are also due to Dr. Thomas Archibald who has read and checked a large part of the final draft of this thesis suggesting various improvements.

Some of the theoretical results in this thesis have been applied to a "real world" model from Edinburgh Petroleum Services Ltd., and it has taken a lot of time and patience to obtain computational results. Here special thanks must be given to Jan Läwen. I would also like to thank the company as a whole for providing a friendly environment.

Last but not least I would like to thank the Department of Mathematics and especially my colleagues in the optimisation group who often helped me with computer and software problems and provided a very friendly atmosphere.

Abstract

In the theory of dynamic programming (DP) the elimination of non-optimal actions is an important topic. For many DP problems the calculation is slow and action elimination helps to speed up the calculation.

A great part of this thesis is dedicated to the development of action elimination procedures for various classes of DP problems. Common to all these action elimination procedures is that they are based on local optimality conditions. Among the classes of DP problems looked at are deterministic allocation problems and stochastic problems with either continuous or discrete state and action spaces.

For DP problems with continuous state and action space the action elimination procedures are based on the Fritz-John first order optimality conditions. For problems with discrete state and action space the action elimination procedures are based on local optimality conditions for discrete problems. It is shown that action elimination based on local optimality conditions usually leads to a speed-up of one order of magnitude.

Chapters 7 and 8 discuss a constrained non-linear oil production optimization problem. In this problem most functions involved are continuous but a few functions contain discontinuities, which seriously undermines the scope of local optimization. A hybrid algorithm combining a dual method, DP and local optimization is proposed and computational results are presented. These results are then compared to those of another hybrid algorithm, which combines Tabu Search and local optimization.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The study of dynamic programming, as we know it today, started at around 1940 and a lot of the initial work had been done by Bellman and Wald. Dynamic programming (DP) can be viewed as recursive programming since common to all dynamic programming optimization problems is a functional optimality equation which leads to a recursive solution method. In numerous papers Bellman identified optimality conditions of optimization problems and through his work on functional equations, dynamic programming and the principle of optimality became well known. Stochastic sequential decision problems are closely linked to stochastic DP. The modern study of stochastic sequential decision problems began with Wald's work on sequential statistical problems during the Second World War. Today the existing theory on DP is vast, especially on stochastic DP which is closely related to Markov decision processes.

Common to all DP formulations are states and actions. Let S be the set of states that a system can occupy and let A be the set of actions that can be taken. When a system is in state $s \in S$ then typically the set of actions that can be taken is $A_s \subset A$. Taking action $a_s \in A_s$ from state s leads to a transition of the system to a new state $\hat{s} \in S$ and incurs a reward (or cost) $r(s, a_s)$. This is the common ingredient of DP problems. However, this is not enough information to define an optimization problem. Different classes of DP problems extend this framework in different ways. For example, in deterministic DP problems $\hat{s}$ is defined by s and a_s, *i.e.* the new state of the system is determined by the old state and by the action that has been taken. In stochastic DP problems $\hat{s}$ is stochastic with the probability distribution depending on s and a_s. For some problems the number of transitions of states is finite, for others it is infinite, again for others the number of transitions is finite with probability one. Also the objective function varies depending on the problem.

For many deterministic, finite horizon (*i.e.* finite number of transitions of

states) DP problems the objective can be written as

$$\text{maximize} \sum_{i=0}^{n-1} r(s_i, a_i) \quad + R(s_n) \tag{1.1}$$

where s_0 is a given initial state, $a_i \in A_{s_i}$, n is the number of transitions, $s_{i+1} = \hat{s}(s_i, a_i)$ and $R(s_n)$ is a terminal reward. The maximization in (1.1) takes place over $\{a_i\}_{i=0}^{n-1}$. Let's assume that the objects S, A_s, r and R are such that the maximum in 1.1 exists.

For many stochastic, finite horizon DP problems the objective can be formulated as

$$\text{maximize } \mathbb{E}\left(\sum_{i=0}^{n-1} r(s_i, a_i) \quad + R(s_n)\right) \tag{1.2}$$

where now s_{i+1} is a random variable with the distribution depending on s_i and on $a_i \in A_{s_i}$. The maximization in (1.2) takes place over all policies of choosing actions, which means that maximization takes place over all functions π such that $a_i = \pi(i, s_i)$. π determines which action is taken when in the i-th decision period the state occupied by the system is s_i.

For many stochastic, infinite horizon DP problems the objective can be written as

$$\text{maximize } \mathbb{E}\left(\sum_{i=0}^{\infty} \alpha^i r(s_i, a_i)\right) \tag{1.3}$$

where $0 < \alpha < 1$ is a discount factor. To ensure convergence, assume that the function r is bounded.

A lot of problems can be written in one of the forms (1.1), (1.2), (1.3) or in one of these forms when they are slightly changed. The DP optimality condition to (1.1) consists of a value function

$$\begin{aligned} F_0(s) &:= R(s), \\ F_j(s) &:= \max_{\{a_i\}_{i=n-j}^{n-1}} \left\{ \sum_{i=n-j}^{n-1} r(s_i, a_i) \quad + R(s_n) \quad : s_{n-j} = s \right\} \quad \text{for } j \geq 1 \end{aligned}$$

and then establishing the functional equation

$$F_j(s) = \max_{a \in A_s} \{r(s, a) + F_{j-1}(\hat{s}(s, a))\}. \tag{1.4}$$

Then the optimal objective value of (1.1) is $F_n(s_0)$ and can be calculated recursively using (1.4). Having found $F_n(s_0)$ the optimal path of states and actions of problem (1.1) are found as follows: s_1 and a_0 are the state and action for which

$$F_n(s_0) = r(s_0, a_0) + F_{n-1}(s_1),$$

where $a_0 \in A_{s_0}$ and $s_1 = \hat{s}(s_0, a_0)$. Similarly s_2 and a_1 are the state and action for which

$$F_{n-1}(s_1) = r(s_1, a_1) + F_{n-2}(s_2),$$

where $a_1 \in A_{s_1}$ and $s_2 = \hat{s}(s_1, a_1)$. And so on ....

For the objective (1.2) of a stochastic, finite horizon problem the value function is

$$\begin{aligned} F_0(s) &:= R(s), \\ F_j(s) &:= \max_{\pi} \left\{ I\!E \left(\sum_{i=n-j}^{n-1} r(s_i, a_i) \quad + R(s_n) \quad \middle| s_{n-j} = s \right) \right\} \qquad \text{for } j \geq 1 \end{aligned}$$

where π is a policy of choosing actions, which means that π is a function and $a_i = \pi(i, s)$. The maximization in the above definition takes place over the set of policies. The functional equation is

$$F_j(s) = \max_{a \in A_s} \{ r(s, a) + I\!E F_{j-1}(\hat{s}(s, a)) \}$$

where $\hat{s}(s, a)$ is a random variable now. The argument a, which achieves the maximum in the RHS, defines $\tilde{\pi}(j, s)$ where $\tilde{\pi}$ is an optimal policy.

For the objective (1.3) of a stochastic DP problem with infinite horizon the value function and functional equation are

$$\begin{aligned} F(s) &:= \max_{\pi} \left\{ I\!E \left(\sum_{i=0}^{\infty} \alpha^i r(s_i, a_i) \quad \middle| s_0 = s \right) \right\}, \\ F(s) &= \max_{a \in A_s} \{ r(s, a) + \alpha I\!E F(\hat{s}(s, a)) \}. \end{aligned}$$

For many DP problems the state space S can be partitioned into subsets S_0, S_1, S_2,... such that when $s \in S_i$ (for some $i \geq 0$) and any action $a \in A_s$ is taken then the system next occupies a state $\hat{s}$ satisfying $\hat{s} \in S_{i+1}$. In such case the problem has decision periods or stages.

For many DP problems the difficulty is identifying suitable states, actions and stages. The next section discusses two introductory examples, the first one is straightforward, the second one has some entertainment value and shows the difficulty of finding states, actions and stages.

1.1 Two introductory examples

1.1.1 Shortest path problem

Consider the shortest path problem of getting from node 7 to node 0 in the acyclic graph of Figure 1.1 where traversing each arc has a specific cost. A suitable

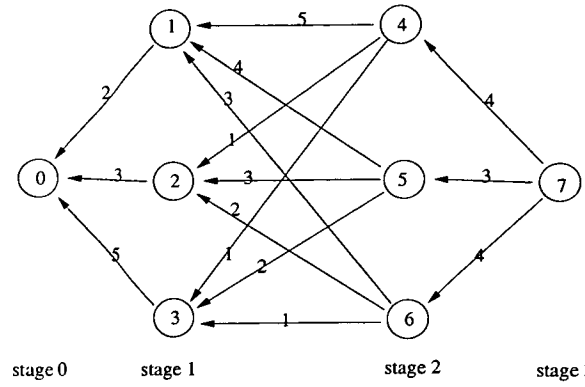

Figure 1.1: A shortest path problem

definition of state is

$$\text{state } i: \text{ getting from node } i \text{ to node } 0.$$

This means every node is associated with a state. A suitable definition of a value function would be

$$F(i) := \text{ the cost of the cheapest way from node } i \text{ to } 0$$

A suitable definition of action to be taken from state i is

$$A(i) := \text{ next node to be visited from node } i \text{ on a cheapest route.}$$

The states of this problem can be partitioned into stages. State 0 is the only state of stage 0, states 1,2,3 belong to stage 1, states 4,5,6 belong to stage 2 and state 7 is the only state of stage 3. The DP optimality equation can be written as

$$F(i) = \min_j \left\{ \begin{array}{c} (\text{cost of going from node } i \text{ to } j) + F(j) \quad : \text{state } j \text{ belongs to} \\ \text{the stage one before that which state } i \text{ belongs to} \end{array} \right\}$$

In words: the optimal route from node i to 0 consists always of a first arc traversal towards node 0 and then to carry on from there on the cheapest possible route to 0. The DP calculation is straightforward, $F(0) = 0$ and having calculated F for all states of stage k the recursion yields F for all states of stage $k+1$. It turns out that F(7)=8 which means that the cheapest route has cost 8. The optimal path can be obtained by working the optimal actions backwards starting from state 7.

If the problem is of the same shape but larger, with stages $0, 1, 2, ..., m, m+1$ and with n nodes in each of stages $1, 2, ..., m$, and if every node is connected to every node of the neighbouring stage, then there are n^m different paths from one end of the network to the other end. The work involved in the DP calculation is $O(mn^2)$, *i.e.* a lot less than total enumeration when m and n are large.

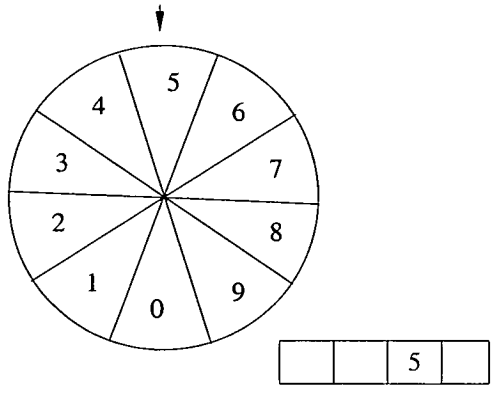

Figure 1.2: A spinning wheel game

1.1.2 A spinning wheel problem

Consider the game, illustrated in Figure 1.2, where a spinning wheel has as possible outcomes the decimal digits 0,1,2,...,9. All possible outcomes have equal probability. The wheel is spun four times and after each spin a player has to put the outcome digit into one of the four boxes. The aim is to have a large four digit number at the end. What is the best strategy for placing the digits? For example, if the first spin returns a 9 the player would (if he or she is sensible) place it into the first box from the left. If at any time a spin returns a 0 the player would place it into the last box which is still free. But what should be done if the first or second spin returns a 6 or 7?

Two DP approaches to this problem will be discussed. The first approach is easier to perceive but less elegant than the second one. A good objective in this game is to achieve a high expectation of the final four digit number. In the first approach let's identify a state as

$$(p_1, p_2, p_3, p_4)$$

where each of p_1, p_2, p_3, p_4 corresponds to one of the objects $\square$, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. A state (p_1, p_2, p_3, p_4) is interpreted as the four places being presently filled in with p_1, p_2, p_3, p_4 from left to right. If $p_i = \square$ then place i contains no digit and is free. The states can be grouped into stages, stage j being the set of those states (p_1, p_2, p_3, p_4) such that exactly j of p_1, p_2, p_3, p_4 are equal to the object $\square$, *i.e.* those states which have exactly j free places left. A suitable value function is

$F(p_1, p_2, p_3, p_4)$:= expectation of the four digit number at the end
given that the game starts with the places being p_1, p_2, p_3, p_4
and given that an optimal placing-strategy is applied.

Let r be the random variable of the result of a spin. The DP optimality equation can be written as

$$F(p_1,p_2,p_3,p_4) = I\!E\Big(\max\Big\{ F(r,p_2,p_3,p_4)I[p_1=\square], \quad F(p_1,r,p_3,p_4)I[p_2=\square], \\ F(p_1,p_2,r,p_4)I[p_3=\square], \quad F(p_1,p_2,p_3,r)I[p_4=\square] \Big\}\Big)$$

where $I[..]$ is the indicator function taking value 1 if the argument is true and 0 otherwise. For states of stage 0 the value of F is simply the number $1000p_1$ $+100p_2$ $+10p_3$ $+p_4$. Knowing F for all states of stage j the value of F can be calculated for all states of stage $j+1$ using the above recursion. $F(\square,\square,\square,\square)$ is the maximal expectation of the four digit number in the spinning wheel game when using an optimal strategy. Doing this DP calculation an optimal placing strategy can be discovered. There are 11^4 states and probably nobody would like to do this DP calculation by hand.

Another DP approach, which allows a solution by hand, is the following: First of all observe that each place can be given a value. The first place has value 10^3, the second one has value 10^2, the third one has value 10^1 and the last has value 1. Let a state be identified by

$$(i, A_1, ..., A_i)$$

where $i = 1,2,3,4$ and $A_1, ..., A_i$ are values of free places with $A_1, ..., A_i$ arranged in decreasing order. For instance, the state $(3, A_1, A_2, A_3)$ corresponds to there being three free places with values A_1, A_2, A_3. Let the value functions be

$$F_i(A_1, ..., A_i) \;:=\; \text{maximal expectation of the end number when a spinning wheel game is played, there are } i \text{ free places and the values of the places are } A_1, ..., A_i.$$

The DP optimality equation is

$$F_i(A_1, .., A_i) = I\!E\Big(\max_{j\in\{1,..,i\}} \Big\{ rA_j + F_{i-1}(B_1, .., B_{i-1}) \quad : B_k = A_k \text{ for } k<j, \\ B_k = A_{k+1} \text{ for } k \geq j \Big\}\Big)$$

where r is the random variable of the result of a spin. Let's outline the DP calculation.

$$F_1(A_1) = I\!E(rA_1) = 4.5A_1$$

because if there is one free place only then there is no choice but to place the first spin in this free place and the expectation of a spin is 4.5. Now consider a

state $(2, A_1, A_2)$. By assumption $A_1 > A_2$. If r is placed in the first place then the expectation of the end number is

$$rA_1 + F_1(A_2) \quad (= rA_1 + 4.5A_2).$$

If r is placed in the second place then the expectation of the end number is

$$F_1(A_1) + rA_2 \quad (= 4.5A_1 + rA_2).$$

Therefore it is optimal to place r in the first place if $r > 4.5$ and in the second place if $r < 4.5$. Then

$$\begin{aligned} F_2(A_1, A_2) &= I\!E\,(F_1(A_1) + rA_2 \quad | r \leq 4)\,\mathrm{P}(r \leq 4) \\ &\quad + I\!E\,(rA_1 + F_1(A_2) \quad | r \geq 5)\,\mathrm{P}(r \geq 5) \\ &= (4.5A_1 + 2A_2)\frac{1}{2} + (7A_1 + 4.5A_2)\frac{1}{2} \\ &= \frac{23}{4}A_1 + \frac{13}{4}A_2 \end{aligned}$$

By looking at the last line one can identify the optimal strategy of placing a spin when three free places are left: If r, the result of a spin, is greater than $\frac{23}{4}$ (*i.e.* if $r = 6, 7, 8, 9$) it is placed in the first free place, if $\frac{23}{4} > r > \frac{13}{4}$ (*i.e.* if $r = 4, 5$) it is placed in the second free place and otherwise it is placed in the last free place. Calculating $F_3(A_1, A_2, A_3)$ in a similar way as $F_2(A_1, A_2)$ has been calculated, one finds

$$F_3(A_1, A_2, A_3) = \frac{129}{20}A_1 + \frac{9}{2}A_2 + \frac{51}{20}A_3$$

From this, one can read off the optimal strategy for where to place the first spin in the spinning wheel game with four free places: if $r = 7, 8, 9$ (*i.e.* $r > \frac{129}{20}$) it is put in the first, if $r = 5, 6$ (*i.e.* $\frac{129}{20} > r > \frac{9}{2}$) it is put in the second, if $r = 3, 4$ (*i.e.* $\frac{9}{2} > r > \frac{51}{20}$) it is put in the third and if $r = 0, 1, 2$ (*i.e.* $\frac{51}{20} > r$) it is put in the last place. One finds

$$F_4(A_1, A_2, A_3, A_4) = \frac{1383}{200}A_1 + \frac{1057}{200}A_2 + \frac{743}{200}A_3 + \frac{417}{200}A_4.$$

With $A_1 = 10^3$, $A_2 = 10^2$, $A_3 = 10$, $A_4 = 1$ the expectation of the four digit number under an optimal strategy in the spinning wheel game turns out to be 7482.735. If the outcomes of the spins were placed at random then the expected value of the four digit number would be 4999.5.

1.2 Motivation for action elimination based on local optimality conditions

The first half of the thesis develops action elimination procedures based on local optimality conditions for various classes of DP problems. In this section the motivation for this is given and difficulties of the implementation are outlined.

Consider the allocation problem

$$\text{maximize} \quad \sum_{i=1}^{r} f_i(x_i) \tag{1.5}$$

$$\text{subject to} \quad \sum_{i=1}^{r} x_i = n, \tag{1.6}$$

$$x_i \in \mathbb{Z}, \quad x_i \geq 0 \quad \text{for all } i \tag{1.7}$$

where $n \in \mathbb{N}$ and f_i are functions $\mathbb{Z} \to \mathbb{R}$. These problems can be done with DP when the states are (j,k) with $j,k \in \mathbb{Z}$, $1 \leq j \leq r$ and $0 \leq k \leq n$. The state (j,k) is the state of allocating k units of the resource among the first j functions f_i. The value function and the optimality conditions are

$$F_j(k) := \max_{\{x_i\}_{i=1}^{j}} \left\{ \sum_{i=1}^{j} f_i(x_i) \quad : \sum_{i=1}^{j} x_i = k, \quad x_i \in \mathbb{Z}, \quad 0 \leq x_i \quad \forall i \right\}, \tag{1.8}$$

$$F_j(k) = \max_{x_j} \left\{ f_j(x_j) + F_{j-1}(k - x_j) \quad : x_j \in \mathbb{Z}, \quad 0 \leq x_j \leq k \right\}. \tag{1.9}$$

The DP calculation initialises $F_1(k) = f_1(k)$ and having calculated $F_{j-1}(k)$ for all relevant k for some j, it calculates $F_j(k)$ using (1.9). Assume without loss of generality (WLOG) that $f_i(x_i) \geq 0$ for all i when $x_i \geq 0$, let M be an upper bound on $f_i(x_i)$ for all i when $x_i \geq 0$ and assume (WLOG) that $f_i(-1) = -rM$ for all i. Let $(\hat{x}_1, \hat{x}_2, .., \hat{x}_r)$ be an optimal allocation for this problem. A necessary optimality condition is

$$\begin{aligned} & \min_{1 \leq i \leq r} \max\{f_i(\hat{x}_i) - f_i(\hat{x}_i - 1), \quad f_i(\hat{x}_i + 1) - f_i(\hat{x}_i)\} \\ \geq \; & \max_{1 \leq i \leq r} \min\{f_i(\hat{x}_i) - f_i(\hat{x}_i - 1), \quad f_i(\hat{x}_i + 1) - f_i(\hat{x}_i)\} \end{aligned} \tag{1.10}$$

This will be derived in a more general form in chapter 5. Condition (1.10) is related to the Kuhn-Tucker conditions for a continuous problem of the form (1.5),(1.6) with differentiable functions f_i. This can be seen by replacing all occurrences of "1" by δx in (1.10) and dividing the whole inequality by δx.

$$\begin{aligned} & \min_{1 \leq i \leq r} \max\left\{ \frac{f_i(\hat{x}_i) - f_i(\hat{x}_i - \delta x)}{\delta x}, \quad \frac{f_i(\hat{x}_i + \delta x) - f_i(\hat{x}_i)}{\delta x} \right\} \\ \geq \; & \max_{1 \leq i \leq r} \min\left\{ \frac{f_i(\hat{x}_i) - f_i(\hat{x}_i - \delta x)}{\delta x}, \quad \frac{f_i(\hat{x}_i + \delta x) - f_i(\hat{x}_i)}{\delta x} \right\} \end{aligned}$$

In the limit as $\delta x \to 0$ this inequality reduces to

$$\min_{1\le i\le r}\left\{\frac{\mathrm{d}f_i}{\mathrm{d}x}(\hat{x}_i)\right\} \quad \ge \quad \max_{1\le i\le r}\left\{\frac{\mathrm{d}f_i}{\mathrm{d}x}(\hat{x}_i)\right\}$$

from which it follows that $\frac{\mathrm{d}f_i}{\mathrm{d}x}(\hat{x}_i) = \lambda$ for all $1 \le i \le r$ for some λ, which are the Kuhn-Tucker conditions for a continuous problem of the form (1.5),(1.6) with differentiable functions f_i.

Going back to the discrete problem (1.5)-(1.7), it will be shown in chapter 5 that if condition (1.10) is not satisfied then there exist indices j and k such that $j \neq k$ and

$$f_j(\hat{x}_j + 1) - f_j(\hat{x}_j) \quad > \quad f_k(\hat{x}_k) - f_k(\hat{x}_k - 1).$$

Then the point $(x_1^*, x_2^*, .., x_r^*)$ with

$$\begin{aligned} x_j^* &= \hat{x}_j + 1, \\ x_k^* &= \hat{x}_k - 1, \\ x_i^* &= \hat{x}_i \quad \text{for all } i \neq j, i \neq k \end{aligned}$$

is feasible and has a higher objective than $(\hat{x}_1, \hat{x}_2, ..., \hat{x}_r)$ since

$$\begin{aligned} \sum_{i=1}^{r} x_i^* &= \sum_{i=1}^{r} \hat{x}_i \quad = \quad n, \\ \sum_{i=1}^{r} f_i(x_i^*) &= \sum_{i\neq j, i\neq k} f_i(x_i^*) \quad + f_j(x_j^*) + f_k(x_k^*) \\ &= \sum_{i\neq j, i\neq k} f_i(\hat{x}_i) \quad + f_j(\hat{x}_j) + (f_j(x_j^*) - f_j(\hat{x}_j)) \\ &\qquad + f_k(\hat{x}_k) + (f_k(x_k^*) - f_k(\hat{x}_k)) \\ &= \sum_{i=1}^{r} f_i(\hat{x}_i) \quad + (f_j(\hat{x}_j + 1) - f_j(\hat{x}_j)) \\ &\qquad -(f_k(\hat{x}_k) - f_k(\hat{x}_k - 1)) \\ &> \sum_{i=1}^{r} f_i(\hat{x}_i), \end{aligned}$$

so $(\hat{x}_1, \hat{x}_2, ..., \hat{x}_r)$ is not locally optimal. Notice that the definition of $F_j(k)$ in (1.8) is itself an optimization problem of the form (1.5)-(1.7). Hence an optimality condition similar to (1.10) must hold for the optimal arguments achieving the maximum in the RHS of (1.8). When $F_j(k)$ is calculated using recursion (1.9), $k+1$ possible values for x_j must be checked. In this thesis it will be shown how to use optimality conditions of the form (1.10) in order to determine $F_j(k)$ with fewer than $k+1$ checks for different values of x_j.

If the domain of x_i in problem (1.5),(1.6) is continuous instead of discrete and if the functions f_i are differentiable is it possible to use first order optimality conditions to speed up the DP process? If on top of that the constraint (1.6) is replaced by

$$\sum_{i=1}^{r} w_i(x_i) = q$$

where w_i are differentiable functions, is it still possible to exploit first order optimality conditions? How about the case when $x_i \in \mathbb{R}^{n_i}$ with $n_i \in \mathbb{N}$ (instead of $x_i \in \mathbb{R}$) and $q \in \mathbb{R}^m, w_i \in \mathbb{R}^m$? Do there exist certain classes of stochastic DP problems for which first order optimality conditions can be exploited?

All of these questions have a positive answer and they are discussed in the first half of this thesis. For a lot of classes of DP problems the complexity and the computational time of DP can be reduced by one order of magnitude by exploiting first order optimality conditions to eliminate actions.

1.3 Statement of the Fritz-John conditions

A central topic in this thesis is the action elimination in DP based on first order optimality conditions. The Fritz-John first order optimality conditions will be used frequently and therefore these conditions should be stated. In the problem

$$\begin{array}{lll} \text{maximize} & f(x) & \\ \text{subject to} & u_i(x) \leq 0 & \text{for } i = 1, 2, .., k \\ & c_i(x) = 0 & \text{for } i = 1, 2, .., m \end{array}$$

let $x \in \mathbb{R}^n$ and let the functions f, u_i, c_i be differentiable functions $\mathbb{R}^n \to \mathbb{R}$. If $\hat{x}$ is a local optimum then there exist dual multipliers $\lambda \in \mathbb{R}^{k+m}$ and $\lambda_0 \in \{0, 1\}$ such that

$$\begin{array}{rcll} \lambda_0 \nabla f(\hat{x}) & = & \sum_{i=1}^{k} \lambda_i \nabla u_i(\hat{x}) \quad + \quad \sum_{i=1}^{m} \lambda_{i+k} \nabla c_i(\hat{x}), & \\ \lambda_i & \geq & 0 \qquad \text{for } i \leq k, & \\ \lambda_i u_i(\hat{x}) & = & 0 \qquad \text{for } i \leq k. & \end{array}$$

When $\lambda_0 = 1$ these conditions are also called Kuhn-Tucker conditions. If the gradients $\nabla u_i(\hat{x})$ of active inequality constraints (*i.e.* for which $u_i(\hat{x}) = 0$) and the gradients of the equality constraints $\nabla c_1(\hat{x}), .., \nabla c_m(\hat{x})$ are linearly independent then the Fritz-John conditions always hold with $\lambda_0 = 1$, *i.e.* they hold as Kuhn-Tucker conditions. The conditions $\lambda_i u_i(\hat{x}) = 0$ for $i \leq k$ are called complementary slackness conditions.

These first order optimality conditions are widely used in optimization theory. A derivation and discussion of these conditions can be found in [7] and [10].

1.4 Motivations for this thesis

This Ph.D started with the task of efficiently solving gaslift allocation problems which arise in the oil production industry. The problems looked at had the following two forms:

$$\max \sum_{h=1}^{r} f_h(x_h) \tag{1.11}$$

$$\text{subject to} \quad \sum_{h=1}^{r} x_h \leq \hat{q}_1, \tag{1.12}$$

where $x_h \in \mathbb{R}$, $x_h \geq 0$, $f_h : \mathbb{R} \to \mathbb{R}$ for $h = 1, ..., r$, and $\hat{q}_1 \geq 0$

or (1.11), (1.12) and the additional constraint

$$\sum_{h=1}^{r} c_h f_h(x_h) \leq \hat{q}_2 \tag{1.13}$$

where the c_h are constants. $f_h(x_h)$ is the oil rate flowing out of well h when the injection rate of gas into that well is x_h. (1.11) is the objective function and (1.12) is the constraint describing the limit on the available total injection gas rate. Usually, the fluid flowing out of a well does not only consist of oil but also of waste (like water, sulphur ...) which has to be separated from the oil and disposed of. Often the waste is proportional to the oil with the proportionality constant depending on the well. (1.13) is a constraint expressing the waste separation capacity. Usually the functions f_h have a discontinuity. (A typical shape for these functions is given in Figure 2.1.) Due to the discontinuities local optimization techniques are not suitable for problems of the form (1.11), (1.12) and in particular for problems which also include (1.13) as a constraint.

These gaslift allocation problems can be solved by DP (which will be shown in chapter 2). The study of methods to speed up the DP calculation of gaslift allocation problems lead to action elimination based on first order optimality conditions.

Then it was realised that this kind of action elimination is more general and it was applied to other classes of DP-problems. (Chapters 5 and 6.)

In the optimization of oil production networks problems of the form (1.11), (1.12), (1.13) are only subproblems. In the last part of the Ph.D a more complicated oil production problem was looked at. This problem also considers compressors, flow through pipes and the associated pressure drop in addition to the

gaslift allocation. The problem was modelled and a solution procedure for it was developed.

1.5 Outline of the following chapters

Chapter 2 introduces the basic idea of how to use first order optimality conditions as the basis for action elimination. The class of problems considered are deterministic allocation problems. Most important in chapter 2 are the sections 2.2, 2.3 and 2.4.

Chapter 3 discusses discretisation issues in combination with FJ action elimination. Section 3.1 presents a discretisation scheme which works well in combination with FJ action elimination and gives good results for allocation problems with a large number of possible allocation destinations. Section 3.2 shows how FJ action elimination can be applied with interpolation DP methods.

Chapter 4 shows difficulties of applying FJ action elimination to problems with multi dimensional state and action spaces such that it results in a speed-up of the DP. Various techniques are discussed to overcome these difficulties.

Chapter 5 looks at deterministic discrete allocation problems. For a class of discrete allocation problems with linear constraints necessary optimality conditions are derived. These optimality conditions are taken as the basis for action elimination.

Chapter 6 discusses certain types of stochastic DP problems which allow action elimination based on first order optimality conditions.

Chapters 2-6 are closely related, they all have in common the action elimination based on local optimality conditions. Chapter 7 and 8 do not relate to the previous chapters except that a subproblem in an algorithm in chapter 8 is solved by DP with FJ action elimination and variable grid discretisation scheme (chapter 3).

Chapter 7 discusses a dual algorithm in a framework such that the results of this chapter can be applied to an industrial problem in chapter 8. Chapter 8 deals with an oil production problem with some functions involved having discontinuities, which seriously undermine the scope of local optimization. An approach combining the dual algorithm of chapter 7 with local optimization is tried in order to overcome the difficulties posed by the discontinuous functions.

Chapter 2

Fritz-John action elimination

2.1 Introduction

Dynamic programming (DP) can be applied to find a global optimum of a wide range of problems, but for many potential applications it is too slow to be of practical use. This chapter introduces a new action elimination method for a class of problems with differentiable objective and constraint functions. The action elimination can reduce the work in the inner computational loop of the DP method and is based on the Fritz-John conditions.

This chapter deals with separable optimization problems of the following form.

$$\max \sum_{h=1}^{r} f_h(x_h) \tag{2.1}$$

$$\text{subject to} \quad \sum_{h=1}^{r} w_h(x_h) \;=\; \hat{q}, \tag{2.2}$$

where $x_h \in \mathbb{R}^{n_h}$, $f_h : \mathbb{R}^{n_h} \to \mathbb{R}$, $w_h : \mathbb{R}^{n_h} \to \mathbb{R}^m$ for $h = 1, ..., r$, and $\hat{q} \in \mathbb{R}^m$.

The problem divides into r stages. The variables are partitioned among the stages, and the objective function and each of the constraint functions is a sum of functions each of which depend only on variables from one stage. Each stage's contribution to the objective and to the constraints is therefore independent of the variables of the other stages.

Many allocation problems have the form given by (2.1) and (2.2). For example, in gaslift allocation problems in oil production optimization x_h is the rate of gaslift to well h, $f_h(x_h)$ is the oil production rate of well h (depending on the gaslift rate x_h) so the objective function (2.1) is the total oil production rate. The constraints (2.2) certainly include the gaslift availability constraint of the form $\sum_h x_h = G$ where G is the total rate of gaslift, sometimes they include a water handling capacity constraint and other constraints. Nishikiori *et al* [12]

give a full description of such problems. Note that in [12] an optimization method is applied which only leads to a local optimum whereas dynamic programming would lead to a global optimum.

The problem (2.1) and (2.2) has a natural formulation as a dynamic programming problem. To derive this formulation first define

$$Q_i = \left\{ q \in I\!R^m : \exists x_h \in I\!R^{n_h} \text{ for } 1 \leq h \leq i \text{ such that } \sum_{h=1}^{i} w_h(x_h) = q \right\},$$

which is the set of values for q for which the constraint (2.4) is feasible, then define the value function F_i for all i, $1 \leq i \leq r$ and $q \in Q_i$, by

Problem $P_i(q)$**:**

$$F_i(q) \equiv \max_{x_1, x_2, \ldots, x_i} \sum_{h=1}^{i} f_h(x_h) \tag{2.3}$$

$$\text{subject to } \sum_{h=1}^{i} w_h(x_h) = q, \tag{2.4}$$

and for $q \notin Q_i$ define $F_i(q) := -\infty$. $F_i(q)$ is the optimal value for a subproblem with i stages and constraint right hand side q. Note that every subproblem is of the same structure as the original problem given in (2.1), (2.2) . With this notation, the optimal value of the original problem is $F_r(\hat{q})$. Furthermore, there is a bijective relation between states (q, i) and subproblems $P_i(q)$. In DP the vector x_i is called the *action* at stage i. For $i > 1$, problem (2.3) and (2.4) can be rewritten as

$$F_i(q) \equiv \max_{x_i} \quad \left\{ f_i(x_i) + \max_{x_1, \ldots, x_{i-1}} \left\{ \sum_{h=1}^{i-1} f_h(x_h) : \sum_{h=1}^{i-1} w_h(x_h) = q - w_i(x_i) \right\} \right.$$
$$\left. : q - w_i(x_i) \in Q_{i-1} \right\},$$

which yields the DP optimality equation

$$F_i(q) = \max_{x_i} \left\{ f_i(x_i) + F_{i-1}(q - w_i(x_i)) : x_i \in I\!R^{n_i} \right\}. \tag{2.5}$$

Note that the above recursion holds not only for $q \in Q_i$ but also for $q \notin Q_i$. The reason for this is that $F_i(q) = -\infty$ for $q \notin Q_i$ by definition and

$$q \notin Q_i \Leftrightarrow (q - w_i(x_i)) \notin Q_{i-1} \quad \forall x_i \in I\!R^{n_i}.$$

A computational scheme for the problem would involve first finding the function F_1 and then using the DP recurrence (2.5) to calculate F_i for stages $i = 2, \ldots, r$

using the previously calculated values of F_{i-1}. By using the definition of Q_1 and of $F_1(q)$ it is found that

$$F_1(q) = -\infty \quad \text{if } w_1(x_1) \neq q \quad \forall x_1 \in I\!R^{n_1},$$
$$F_1(q) = \max_{x_1 \in I\!R^{n_1}} \{f_1(x_1) : w_1(x_1) = q\} \quad \text{if } \exists x_1 \in I\!R^{n_1} \text{ with } w_1(x_1) = q.$$

Except for problems with very special structure the functions F_i cannot be found exactly and so must be approximated. The most common approach is to replace the continuum of values for q by a finite number of discrete values and also to replace each continuous optimization problem for the optimal action x_h at stage h, by a search over a finite number of discrete values. This approach has the advantage of being able to find an approximate global optimum for problems where the functions f_h and w_h are arbitrary piecewise smooth functions, however except for problems where m and all the n_h are small it is computationally intractable.

Historically, Bellman's book [1] was the origin for the research area of dynamic programming as we know it today. Gilmore and Gomory [9] were the first to use action elimination for the labelling (or reaching) method, which will be described in section 2.3, by exploiting special structure in knapsack models. The action space in such models is discrete. Lagrange multipliers have been known for two centuries, but they were used only for continuous problems in the beginning. Everett [5] suggested their use for problems with integer variables and Fox and Landi [8] used them in dynamic programming. Most of their examples are discrete and the conjunction with Fritz John conditions or the corresponding optimality conditions in the theory of subdifferentials are not given.

In the next two sections problems are considered in which the functions f_h and w_h are differentiable. A new method is presented that uses the Fritz-John (FJ) optimality conditions for *action elimination*, *i.e.* to reduce the number of actions x_i which have to be examined in the maximization step (2.5) at each stage. Thereafter, the theory will be extended and the new algorithm modified to allow general functions f_h and w_h. For suitable problems this action elimination method reduces the solution time significantly. The complexity of DP is reduced by at least one order of magnitude, with the precise reduction depending on the structure of the specific problem. This chapter provides a general introduction to this new method. There are many possible ways of implementing the method and details are given of the implementation of a few example problems. For some of these a comparison of solution times for DP with and without the FJ action elimination is given.

2.2 General Theory of the FJ Action Elimination

In this section the FJ conditions are stated for problem (2.1) and (2.2) in the case when all the functions are differentiable, and it is shown how to use the FJ conditions for action elimination in DP.

Let $\lambda_0 \in I\!R$ and $\lambda \in I\!R^m$ be the multipliers associated with the objective function and constraints of problem (2.3) and (2.4). Further let ∇f_h be the $n_h \times 1$ gradient matrix of f_h and ∇w_h denote the $m \times n_h$ Jacobian matrix of w_h. The Fritz-John (FJ) conditions for problem (2.3) and (2.4) are

$$\lambda_0(\nabla f_h(x_h))^T = \lambda^T \nabla w_h(x_h), \text{ for } h = 1, ..., i, \tag{2.6}$$

$$\sum_{h=1}^{i} w_h(x_h) = q, \tag{2.7}$$

$$\lambda_0 = 0 \text{ or } \lambda_0 = 1, \tag{2.8}$$

$$(\lambda_0, \lambda) \neq 0 \tag{2.9}$$

The FJ conditions are necessary conditions for local optimality. If $\lambda_0 = 1$ they become the Kuhn-Tucker (KT) conditions, which hold at any optimal solution at which the constraint gradients are linearly independent. (See [7], chapter 9 or [10] Chapter 4.)

For given state (q, i) and vector of actions $\boldsymbol{x}_i$ where $\boldsymbol{x}_i^T = (x_1^T, x_2^T, ..., x_i^T)$, let $B_i(q, \boldsymbol{x}_i)$ be the set of all vectors (λ_0, λ) with $\lambda_0 \in I\!R$, $\lambda \in I\!R^m$ such that (λ_0, λ) and $\boldsymbol{x}_i$ satisfy the FJ-conditions (2.6)-(2.9). Let $\boldsymbol{x}_{i-1}$ be like $\boldsymbol{x}_i$ except that the last component is missing, *i.e.* $\boldsymbol{x}_{i-1}^T = (x_1^T, x_2^T \ldots x_{i-1}^T)$. Now notice that for $i \geq 2$ the set of properties in (2.6)-(2.9) defining the set $B_{i-1}(q - w_i(x_i), \boldsymbol{x}_{i-1})$ is fully contained in the set of properties defining $B_i(q, \boldsymbol{x}_i)$. Furthermore the properties which are amongst those defining $B_i(q, \boldsymbol{x}_i)$ but not amongst those defining $B_{i-1}(q - w_i(x_i), \boldsymbol{x}_{i-1})$ are

$$\lambda_0(\nabla f_i(x_i))^T = \lambda^T \nabla w_i(x_i) \quad . \tag{2.10}$$

Define $N_i(x_i)$ by

$$N_i(x_i) := \{(\lambda_0, \lambda) : \lambda_0 \in \{0, 1\}, \lambda \in I\!R^m, (2.10)\text{is satisfied}\}$$

With this definition and the previous observations, it follows that for $i \geq 2$

$$B_i(q, \boldsymbol{x}_i) = B_{i-1}(q - w_i(x_i), \boldsymbol{x}_{i-1}) \cap N_i(x_i) \tag{2.11}$$

In the proposed algorithm the above equation will be used to define (λ_0, λ) sets for states recursively.

2.3 Action elimination using the Fritz-John conditions

The most common way to implement dynamic programming is generally known as recursive fixing [3], [4]. This is illustrated by the program below which shows the step of going from stage $i-1$ to stage i for $i \geq 2$. Let $A_i(q)$ denote the optimal action for state (q, i).

for all $q \in I\!R^m$ do
  $F_i(q) := -\infty$
  for all $x_i \in I\!R^{n_i}$ do
    if $F_i(q) < F_{i-1}(q - w_i(x_i)) + f_i(x_i)$ then
      $F_i(q) := F_{i-1}(q - w_i(x_i)) + f_i(x_i)$
      $A_i(q) := x_i$
    end if
  end do x_i
end do q

The outer loop cycles over all states at stage i and the inner loop finds the optimal action for the current state being considered in the outer loop.
The following alternative implementation is generally referred to as reaching or the labelling procedure (See [3], [4]).

for all $q \in I\!R^m$ set $F_i(q) := -\infty$
for all $q \in I\!R^m$ do
  for all $x_i \in I\!R^{n_i}$ do
    if $F_i(q + w_i(x_i)) < F_{i-1}(q) + f_i(x_i)$ then
      $F_i(q + w_i(x_i)) := F_{i-1}(q) + f_i(x_i)$
      $A_i(q + w_i(x_i)) := x_i$
    end if
  end do x_i
end do q

In the recursive fixing method the outer loop fixes a state (q, i) of stage i and the inner loop is used to determine $F_i(q)$. In the labelling method the outer loop fixes a state $(q, i-1)$ of stage $i-1$ and the inner loop does all comparisons of the dynamic programming calculation, in which $F_{i-1}(q)$ is involved. The difference between the two methods is that the nested loops are swapped.

When $F_{i-1}(q) = -\infty$ then all the DP comparisons of the form

$$F_i(q + w_i(x_i)) < F_{i-1}(q) + f_i(x_i)$$

are superfluous. This observation can be exploited within the labelling method to reduce computation. The second line of the above procedure can simply be replaced with:

$$\text{for all } q \in I\!R^m \text{ for which } F_{i-1}(q) > -\infty \text{ do}$$

In the future this modification will always be implemented where it is appropriate.

Since the state space in this description is infinite, the above loops are actually infinite loops. In practice a discretisation has to be introduced to make the sets of states and actions finite.

It is now shown how the action elimination takes place. The new algorithm will employ a function of states $B_i(q)$ defined by

$$B_i(q) = B_i(q, \widetilde{\boldsymbol{x}}_i),$$

where $\widetilde{\boldsymbol{x}}_i$ is the optimal vector for subproblem (q, i) that the algorithm has computed at the time of defining $B_i(q)$. By combining (2.11) and (2.10) we can replace (2.5) with

$$F_i(q) = \max_{x_i \in I\!R^{n_i}} \{f_i(x_i) + F_{i-1}(q - w_i(x_i)) : \exists(\lambda_0, \lambda) \in B_{i-1}(q - w_i(x_i)) \text{ such that } (\lambda_0, \lambda), x_i \text{ satisfy (2.10)}\} \quad (2.12)$$

Replacing (2.5) with (2.12) is the essence of the action elimination!

In order to implement (2.12) efficiently the labelling procedure is used for going from stage $i-1$ to stage i (for $i \geq 2$), as shown in the following procedure.

Procedure 2.3.1

for all $q \in I\!R^m$ set $F_i(q) := -\infty$
for all $q \in I\!R^m$ for which $F_{i-1}(q) > -\infty$ do
  for all $x_i \in I\!R^{n_i}$ for which $\exists(\lambda_0, \lambda) \in B_{i-1}(q)$
      such that $(\lambda_0, \lambda), x_i$ satisfy (2.10) do
    if $F_i(q + w_i(x_i)) < F_{i-1}(q) + f_i(x_i)$ then
      $F_i(q + w_i(x_i)) := F_{i-1}(q) + f_i(x_i)$
      $A_i(q + w_i(x_i)) := x_i$
      $B_i(q + w_i(x_i)) := B_{i-1}(q) \cap N_i(x_i)$
    end if
  end do x_i
end do q

Note that for the FJ action elimination to be efficient there must be a quick way to identify those $x_i \in \mathbb{R}^{n_i}$ which satisfy (2.10) for any given $(\lambda_0, \lambda) \in B_{i-1}(q)$. If the amount of work required to do that is not less than running through all $x_i \in \mathbb{R}^{n_i}$, then the FJ action elimination is not useful.

Next, it will be explained why in the action elimination process it is sufficient to choose $B_i(q) = B_i(q, \widetilde{\boldsymbol{x}}_i)$ where $\widetilde{\boldsymbol{x}}_i$ is any optimal vector for subproblem (q, i). Suppose that $\widetilde{\boldsymbol{x}}_i$ is an optimal vector for subproblem (q, i) and is given by $\widetilde{\boldsymbol{x}}_i^T = (\tilde{x}_1^T, \tilde{x}_2^T, \ldots, \tilde{x}_i^T)$. Further suppose that $\widehat{\boldsymbol{x}}$ is an optimal vector to the (original whole) problem (2.1) and (2.2) and is given by $\widehat{\boldsymbol{x}}^T = (\hat{x}_1^T, \hat{x}_2^T, \ldots, \hat{x}_r^T)$ and its optimal path passes through state (q, i). Then by the DP-optimality equation the vector $\boldsymbol{x}^*$ given by

$$\boldsymbol{x}^{*T} = (\tilde{x}_1^T, \tilde{x}_2^T, \ldots, \tilde{x}_i^T, \hat{x}_{i+1}^T, \hat{x}_{i+2}^T, \ldots, \hat{x}_r^T)$$

is also optimal for the (original whole) problem (2.1) and (2.2). But the action elimination process with $B_i(q) = B_i(q, \widetilde{\boldsymbol{x}}_i)$ does cover the actions necessary to discover $\boldsymbol{x}^*$ as an optimal vector for problem (2.1) and (2.2). Hence result.

2.4 Extension to general functions

In this section the method of the previous section is extended to allow general functions f_h, w_h and to allow x_h to be defined over a general set. When D_h is a set then let the boundary of D_h be denoted by ∂D_h.

Theorem 2.4.1 *Consider the problem (2.1) and (2.2) with the additional constraints that $x_h \in D_h$ for all h where $D_h \subset \mathbb{R}^{n_h}$, and let $\widetilde{\boldsymbol{x}}$ be a global solution to the problem where $\widetilde{\boldsymbol{x}}^T = (\tilde{x}_1^T, \tilde{x}_1^T, \ldots, \tilde{x}_r^T)$. Let I be the set of indices such that for $h \in I$, f_h and w_h are differentiable at $\tilde{x}_h$ and $\tilde{x}_h \notin \partial D_h$. Then there exists (λ_0, λ) such that*

$$\begin{aligned} \lambda_0(\nabla f_h(\tilde{x}_h))^T &= \lambda^T \nabla w_h(\tilde{x}_h) \quad \text{for all } h \in I, \\ \lambda_0 \in \{0, 1\} \quad , \quad \lambda \in \mathbb{R}^m, &\quad (\lambda_0, \lambda) \neq 0. \end{aligned} \tag{2.13}$$

Proof:
Define

$$q^* := \hat{q} - \sum_{h \notin I} w_h(\tilde{x}_h)$$

Then consider the problem

$$\max \sum_{h \in I} f_h(x_h) \tag{2.14}$$

$$\text{subject to} \quad \sum_{h \in I} w_h(x_h) \;=\; q^*, \tag{2.15}$$

$$x_h \in D_h \quad \forall h \in I. \tag{2.16}$$

This problem is solved by $x_h = \tilde{x}_h$ for $h \in I$ since $\tilde{\boldsymbol{x}}$ solves (2.1),(2.2) with the additional constraint that $x_h \in D_h \; \forall h$. Therefore the point $x_h = \tilde{x}_h$ for $h \in I$ is a local solution. Furthermore, w_h and f_h are differentiable at $x_h = \tilde{x}_h$ for $h \in I$. Since (2.14)-(2.16) are of the same form as (2.1)-(2.2) there exists (λ_0, λ) such that the FJ-conditions (2.6)-(2.9) applied to problem (2.14)-(2.16) are fulfilled when $x_h = \tilde{x}_h$ for all $h \in I$. □.

If f_h or w_h have discontinuities then it is possible that problem (2.1)-(2.2) does not have a maximum but a supremum. If $\tilde{x}_h$ is a point at which f_h or w_h is discontinuous then the FJ conditions can not be used to eliminate $\tilde{x}_h$. In the DP recursion it is necessary to always include $\tilde{x}_h$ and a neighbourhood around $\tilde{x}_h$, *i.e.* to always check actions corresponding to $\tilde{x}_h$ and a neighbourhood. The action elimination DP algorithm will now be presented for the case that the functions f_h, w_h have some non-differentiable points. We use the optimality condition from the last theorem. The procedure of going from stage $i-1$ to stage i for $i \geq 2$ is:

$S_i := \{y \in D_i : w_i \text{ or } f_i \text{ is not differentiable at } y\}$
$\overline{S}_i := \{x \in D_i : \exists y \in S_i \text{ such that } |x - y| < \epsilon\}$
for all $q \in I\!R^m$ set $F_i(q) := -\infty$
for all $q \in I\!R^m$ with $F_{i-1}(q) > -\infty$ do
  for all $x_i \in D_i \backslash (\partial D_i \cup \overline{S}_i)$ for which $\exists (\lambda_0, \lambda) \in B_{i-1}(q)$
      such that $(\lambda_0, \lambda), x_i$ satisfy (2.10) do
    if $F_i(q + w_i(x_i)) < F_{i-1}(q) + f_i(x_i)$ then
      $F_i(q + w_i(x_i)) := F_{i-1}(q) + f_i(x_i)$
      $A_i(q + w_i(x_i)) := x_i$
      $B_i(q + w_i(x_i)) := B_{i-1}(q) \cap N_i(x_i)$
    end if
  end do x_i
  for all $x_i \in \partial D_i \cup \overline{S}_i$ do
    if $F_i(q + w_i(x_i)) < F_{i-1}(q) + f_i(x_i)$ then
      $F_i(q + w_i(x_i)) := F_{i-1}(q) + f_i(x_i)$
      $A_i(q + w_i(x_i)) := x_i$
      $B_i(q + w_i(x_i)) := B_{i-1}(q)$

end if

end do x_i

end do q

The purpose of defining $\overline{S}_i$ is to have actions on every side of a discontinuous point $x \in S_i$. In the definition of $\overline{S}_i$ the parameter ϵ is positive and arbitrarily small. Of course, the more non-differentiable points the functions w_i, f_i have and the more points there are on the boundary of D_i, the less effective is the action elimination.

2.4.1 Practical Implementation

For most problems it is only practical to implement the action elimination if the set $B_{i-1}(q)$ has only one element. Usually, if Problem $P_{i-1}(q)$ is feasible the set $B_{i-1}(q)$ has one element or an infinite number of elements. If Problem $P_{i-1}(q)$ is infeasible then $B_{i-1}(q)$ is the empty set. If $B_{i-1}(q)$ has one element and x_i is such that $(\lambda_0, \lambda), x_i$ satisfy (2.10) where $(\lambda_0, \lambda) \in B_{i-1}(q)$ then

$$B_{i-1}(q) \subset N_i(x_i)$$

$$\text{and hence} \qquad B_{i-1}(q) \cap N_i(x_i) = B_{i-1}(q).$$

The following procedure of going from stage $i-1$ to stage i (for $i \geq 2$) only makes use of the action elimination when the set $B_{i-1}(q)$ has one element. It is based on the previous procedure and the above observations about $B_{i-1}(q)$.

$S_i := \{y \in D_i : w_i \text{ or } f_i \text{ is not differentiable at } y\}$

$\overline{S}_i := \{x \in D_i : \exists y \in S_i \text{ such that } |x - y| < \epsilon\}$

for all $q \in I\!R^m$ set $F_i(q) := -\infty$

for all $q \in I\!R^m$ for which $F_{i-1}(q) > -\infty$ do

if $B_{i-1}(q)$ has exactly one element then do

$J := \{x : x \in D_i \backslash S_i, (\lambda_0, \lambda) \in B_{i-1}(q) \text{ and } x \text{ satisfy (2.10)}\}$

for all $x_i \in \overline{S}_i \cup \partial D_i \cup J$ do

if $F_i(q + w_i(x_i)) < F_{i-1}(q) + f_i(x_i)$ then

$F_i(q + w_i(x_i)) := F_{i-1}(q) + f_i(x_i)$

$A_i(q + w_i(x_i)) := x_i$

$B_i(q + w_i(x_i)) := B_{i-1}(q)$

end if

end do x_i

else (if $B_{i-1}(q)$ has more than one element)
for all $x_i \in D_i$ do
if $F_i(q + w_i(x_i)) < F_{i-1}(q) + f_i(x_i)$ then
$F_i(q + w_i(x_i)) := F_{i-1}(q) + f_i(x_i)$
$A_i(q + w_i(x_i)) := x_i$
if $x_i \notin \overline{S}_i \cup \partial D_i$ then
$B_i(q + w_i(x_i)) := B_{i-1}(q) \cap N_i(x_i)$
else
$B_i(q + w_i(x_i)) := B_{i-1}(q)$
end if
end if
end do x_i
end if
end do q

Most often the FJ optimality conditions hold as Kuhn-Tucker conditions. *i.e.* most often the dual multipliers (λ_0, λ) satisfy $\lambda_0 = 1$. The efficient calculation of the set J (in line 6 of the previous procedure) usually involves a considerable amount of coding. Moreover, often the two cases $\lambda_0 = 0$ and $\lambda_0 = 1$ for $(\lambda_0, \lambda) \in B_{i-1}(q)$ have to be considered separately. The amount of coding necessary to calculate J when $\lambda_0 = 0$ is not worth the effort for many applications. The reason for this is that $\lambda_0 = 0$ can only be if the gradients of the constraint functions are linearly dependent at the optimal solution point. (This is a property of the FJ-conditions.) In many applications the number of variables is larger than the number of constraints, and the constraint functions are all of a different form. But then it is very unlikely that the gradients of the constraint functions are linearly dependent at the optimal solution point.

Another practical implementation of going from stage $i-1$ to stage i (for $i \geq 2$) only makes use of the action elimination when the set $B_{i-1}(q)$ has one element (λ_0, λ) and for this element $\lambda_0 = 1$. This amounts to changing line 5 of the previous procedure to:

if $B_{i-1}(q)$ has exactly one element (λ_0, λ) and $\lambda_0 = 1$ then do

2.5 Inequality constraints

In this section problem (2.1), (2.2) with the additional constraints $x_h \in D_h \quad \forall h$ is considered where the equality constraints are replaced by inequality constraints. *i.e.* the problem considered is

$$\max \sum_{h=1}^{r} f_h(x_h) \tag{2.17}$$

$$\text{subject to} \quad \sum_{h=1}^{r} w_h(x_h) \leq \hat{q}, \tag{2.18}$$

$$x_h \in D_h \qquad \forall h, \tag{2.19}$$

where $D_h \subset \mathbb{R}^{n_h}$, $f_h : \mathbb{R}^{n_h} \to \mathbb{R}$, $w_h : \mathbb{R}^{n_h} \to \mathbb{R}^m$ for $h = 1, ..., r$, and $\hat{q} \in \mathbb{R}^m$.

There are two approaches to deal with this problem using FJ-DP.

Approach A:
Apply FJ-DP to problem (2.1), (2.2) with the additional constraint (2.19) as described in the previous sections. At the end scan the value function F_r to find the solution to (2.17)-(2.19). *i.e.* the optimal objective value to (2.17)-(2.19) is

$$\max_q \{F_r(q) : q \leq \hat{q}\}$$

where the inequality sign between the two vectors q and $\hat{q}$ is used component-wise *i.e.* the i-th component of q is less than or equal to the i-th component of $\hat{q}$ for every $i : 1 \leq i \leq m$.

Approach B:
The subproblem corresponding to state (q, i) which defines $F_i(q)$ is changed. $F_i(q)$ is now defined by (2.3),(2.4) but with the equality sign in (2.4) being changed into a '$\leq$' sign and the additional constraint (2.19). The DP-recursion (2.5) still holds. The Fritz John conditions for subproblem (q, i) are

$$\lambda_0 = 0 \text{ or } \lambda_0 = 1, \tag{2.20}$$

$$(\lambda_0, \lambda) \neq 0 \tag{2.21}$$

$$\lambda \geq 0 \tag{2.22}$$

$$\lambda^T (\sum_{h=1}^{i} w_h(x_h) - q) = 0, \tag{2.23}$$

for $h = 1, ..., i$, if f_h, w_h are differentiable at x_h and $x_h \notin \partial D_h$ then

$$\lambda_0(\nabla f_h(x_h))^T = \lambda^T \nabla w_h(x_h), \tag{2.24}$$

where ∇f_h is the $n_h \times 1$ gradient matrix of f_h and ∇w_h denotes the $m \times n_h$ Jacobian matrix of w_h. For given state (q,i) and vector of actions $\boldsymbol{x}_i$ where $\boldsymbol{x}_i^T = (x_1^T, x_2^T, ..., x_i^T)$, $B_i(q, \boldsymbol{x}_i)$ now is the set of all vectors (λ_0, λ) with $\lambda_0 \in \mathbb{R}$, $\lambda \in \mathbb{R}^m$ such that (λ_0, λ) and $\boldsymbol{x}_i$ satisfy the FJ-conditions (2.20)-(2.24). The FJ-DP algorithm now employs a function of states $B_i(q)$ defined by

$$B_i(q) = B_i(q, \widetilde{\boldsymbol{x}}_i),$$

where $\widetilde{\boldsymbol{x}}_i$ is the optimal vector for subproblem (q,i) that the algorithm has computed at the time of defining $B_i(q)$. An FJ-DP algorithm can be applied to problem (2.17)-(2.19) which imposes these modified definitions of $F_i(q)$ and $B_i(q)$ in the initialisation step (*i.e.* when defining $F_1(q), B_1(q)\ \forall q$) and then uses any one of the procedures of sections 2.4 and 2.4.1 for going from stage $i-1$ to stage i.

Both approaches A and B work and solve problem (2.17)-(2.19). Approach B uses stricter FJ conditions than approach A since it has the additional condition (2.22). *i.e.* Approach B uses FJ conditions which have the potential to eliminate more non optimal actions. However, in comparison to approach A approach B defines $F_i(q) > -\infty$ for more states (q,i). This is bad since more work has to be done to go from stage i to stage $i+1$ when there are more states (q,i) of stage i for which $F_i(q) > -\infty$. This more work of approach B can even be of one order of magnitude. For example, if $m = 2$ (*i.e.* there are two constraints) and $n_1 = 1$ (*i.e.* $x_1 \in \mathbb{R}$) then approach A defines $F_1(q) > -\infty$ for a one dimensional subspace of states in the two dimensional state space of stage 1, whereas approach B defines $F_1(q) > -\infty$ for a two dimensional subspace of states in the two dimensional state space of stage 1.

Let's summarise the comparative advantages of approaches A and B. Approach A defines $F_i(q) > -\infty$ for fewer states (q,i), approach B uses stricter FJ conditions and therefore has the potential of eliminating more non-optimal actions. It is possible to design a method which has both of these good features of approaches A and B. In the sequel this method is described.

Approach C:

The definition of $B_i(q)$ is as in approach B. The definition of $F_i(q)$ is, in some sense, a blend of that in approach A and that in approach B. Precisely: $F_i(q)$ is the optimal objective value of

$$\max_{x_1, x_2, ..., x_i} \sum_{h=1}^{i} f_h(x_h) \tag{2.25}$$

$$\text{subject to} \sum_{h=1}^{i} w_h(x_h) \leq q, \tag{2.26}$$

$$x_h \in D_h \qquad \text{for } h = 1, ..., i \tag{2.27}$$

if there exist vectors $x_1, x_2, ..., x_i$ which solve problem (2.25)-(2.27) and for which (2.26) holds as equality (in every component), if such vectors do not exist then $F_i(q) := -\infty$. For this definition of $F_i(q)$ the DP recursion (2.5) holds if $F_i(q) > -\infty$ but fails in general if $F_i(q) = -\infty$. With these definitions of $F_i(q)$ and $B_i(q)$ apply an FJ-DP algorithm to problem (2.1),(2.2) and the additional constraint $x_h \in D_h \ \forall h$ which uses any one of the procedures of section 2.4 or 2.4.1 plus the following steps

for all $q \in I\!R^m$ do
  if $F_i(q) > -\infty$ then
    if $\exists \tilde{q} < q$ with $F_i(\tilde{q}) > F_i(q)$ then $F_i(q) := -\infty$
  end if
end do

added at the end when going from stage $i-1$ to stage i. A procedure of section 2.4 or 2.4.1 for going from stage $i-1$ to stage i basically implements the recursion (2.5) but (2.5) only holds when $F_i(q) > -\infty$, as mentioned before. Hence, in general a procedure of section 2.4 or 2.4.1 calculates $F_i(q)$ correctly if $F_i(q) > -\infty$ and incorrectly in general if $F_i(q) = -\infty$. Adding the above steps to a procedure of section 2.4 or 2.4.1 insures that $F_i(q)$ is also calculated correctly when $F_i(q) = -\infty$. The specific definitions of $F_i(q)$ and $B_i(q)$ must be imposed in the initialisation step *i.e.* when defining $F_1(q), B_1(q) \ \forall q$. At the end of this FJ-DP algorithm scan the value function F_r to find the solution to problem (2.17)-(2.19) *i.e.* the optimal objective value to problem (2.17)-(2.19) is

$$\max_q \{F_r(q) : q \leq \hat{q}\}$$

The optimal distribution vectors $\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_r$ which solve this problem are found by working the actions $A_i(q)$ backwards along the optimal path of states starting from state (q^*, r) where

$$q^* = \arg\max_q \{F_r(q) : q \leq \hat{q}\}.$$

2.5.1 Problems with equality and inequality constraints

Next a problem with inequality and equality constraints will be considered and an efficient FJ-DP method will be outlined. The equality constraints are dealt with

as in previous sections and the inequality constraints are dealt with according to approach C of the previous section. Consider:

$$\max \sum_{h=1}^{r} f_h(x_h) \tag{2.28}$$

$$\text{subject to} \quad \sum_{h=1}^{r} w_h(x_h) \leq \hat{q}, \tag{2.29}$$

$$\sum_{h=1}^{r} v_h(x_h) = \hat{z}, \tag{2.30}$$

$$x_h \in D_h \qquad \forall h, \tag{2.31}$$

where $D_h \subset I\!R^{n_h}$, $f_h : I\!R^{n_h} \to I\!R$, $w_h : I\!R^{n_h} \to I\!R^{m_1}$, $v_h : I\!R^{n_h} \to I\!R^{m_2}$ for $h = 1, ..., r$, and $\hat{q} \in I\!R^{m_1}$, $\hat{z} \in I\!R^{m_2}$.

The definition of $F_i(q, z)$ to be used in the FJ-DP method is the following: $F_i(q, z)$ is the optimal objective value of

$$\max_{x_1, x_2, ..., x_i} \sum_{h=1}^{i} f_h(x_h) \tag{2.32}$$

$$\text{subject to} \quad \sum_{h=1}^{i} w_h(x_h) \leq q, \tag{2.33}$$

$$\sum_{h=1}^{i} v_h(x_h) = z, \tag{2.34}$$

$$x_h \in D_h \qquad \text{for } h = 1, ..., i \tag{2.35}$$

if there exist vectors $x_1, x_2, ..., x_i$ which solve problem (2.32)-(2.35) and for which (2.33) holds as equality, if such vectors do not exist then $F_i(q, z) = -\infty$. For this definition the DP recursion

$$F_i(q, z) = \max_{x_i \in D_i} \{f_i(x_i) + F_{i-1}(q - w_i(x_i), z - v_i(x_i))\} \tag{2.36}$$

holds if $F_i(q, z) > -\infty$ and does not hold in general if $F_i(q, z) = -\infty$. For given state (q, z, i) and vector of actions $\boldsymbol{x}_i$ where $\boldsymbol{x}_i^T = (x_1^T, x_2^T, ..., x_i^T)$, $B_i(q, z, \boldsymbol{x}_i)$ is the set of all vectors $(\lambda_0, \lambda, \mu)$ with $\lambda_0 \in I\!R$, $\lambda \in I\!R^{m_1}$, $\mu \in I\!R^{m_2}$ such that $(\lambda_0, \lambda, \mu)$ and $\boldsymbol{x}_i$ satisfy the FJ-conditions (2.20),(2.22), (2.23),(2.34),

$$(\lambda_0, \lambda, \mu) \neq 0 \text{ and}$$

for $h = 1, ..., i$, if f_h, w_h, v_h are differentiable at x_h and $x_h \notin \partial D_h$ then

$$\lambda_0(\nabla f_h(x_h))^T = \lambda^T \nabla w_h(x_h) + \mu^T \nabla v_h(x_h)$$

where ∇f_h is the $n_h \times 1$ gradient matrix of f_h, ∇w_h denotes the $m_1 \times n_h$ Jacobian matrix of w_h and ∇v_h denotes the $m_2 \times n_h$ Jacobian matrix of v_h. The FJ-DP algorithm now employs a function of states $B_i(q,z)$ defined by

$$B_i(q,z) = B_i(q,z,\widetilde{\boldsymbol{x}}_i),$$

where $\widetilde{\boldsymbol{x}}_i$ is the optimal vector for subproblem (q,z,i) that the algorithm has computed at the time of defining $B_i(q,z)$. These definitions of $F_i(q,z)$ and $B_i(q,z)$ must be imposed when defining $F_1(q,z)$, $B_1(q,z)$ $\forall(q,z)$ initially. When going from stage $i-1$ to stage i for $i \geq 2$ a procedure similar to one of those in section 2.4 or 2.4.1 is used to implement (2.36). At the end of the implementation of (2.36) if for a state (q,z,i) $F_i(q,z) > -\infty$ and $\exists \tilde{q} < q$ such that $F_i(\tilde{q},z) > F_i(q,z)$ then one must redefine $F_i(q,z) := -\infty$ in order to enforce the correct definition of $F_i(q,z)$. The optimal objective value to problem (2.28)-(2.31) is

$$\max_q \{F_r(q,\hat{z}) : q \leq \hat{q}\}.$$

2.6 A single constraint example

In this section an example, which is similar in form to the gaslift allocation problem mentioned in the introduction, will be shown in order to illustrate the details. This example is highly non-linear. The theory in the previous sections was for continuous variables. In practice, however, a discretisation must be introduced. The discretisation which will be used is one to approximate the continuous implementation of FJ-DP. The finer the discretisation is, the smaller the error of the approximated solution will be.

Consider the problem:

$$\begin{aligned} \text{maximize} f(\boldsymbol{x}) := \sum_{h=1}^{r} f_h(x_h) & \\ \text{subject to} \quad \sum_{h=1}^{r} x_h &= \hat{q}, \\ x_h \in I\!R, \quad x_h \in [0,d_h] & \qquad 1 \leq h \leq r, \end{aligned} \tag{2.37}$$

where $r = 50$ and $\hat{q} = 1000$. The functions f_i of the example are characterised by four numbers a_i, b_i, c_i, d_i in the following way:

$$f_i(x) = \begin{cases} 0 & \text{if } 0 \leq x < c_i, \\ \log(a_i + x) + b_i & \text{if } c_i \leq x \leq d_i, \end{cases} \tag{2.38}$$

The numbers a_i, b_i, c_i, d_i defining the functions f_i are given in Table 2.1.

i	a_i	b_i	c_i	d_i	i	a_i	b_i	c_i	d_i
1	−9	20	11	42	26	−18	2	20.7	46
2	−18	20	20	41	27	−8	4.2	10.2	30.9
3	−8	40	10	34	28	−8	5.1	10.7	49
4	−8	40	13	40	29	−16	2.9	20	46.8
5	−16	2	19	40.5	30	−5	3.4	10	34
6	−5	4	11.5	32.8	31	−9	5	15	40
7	−9	5	15.7	41.9	32	−17.1	2	25	40
8	−18	3.7	25	49	33	−8	4	15.3	30.9
9	−8	4	15.2	37.4	34	−8.9	5	20.3	40.9
10	−8	5	20.9	39	35	−16	3.2	30.7	50
11	−16	2.9	26.8	40.2	36	−5	3.8	15.3	30
12	−5	4	15	30.3	37	−9	5	27.2	47.9
13	−9	5	28	40.2	38	−18	2.7	30	36.8
14	−18	2.8	29.5	48.7	39	−8	4.2	24.6	30
15	−8	4	25	30	40	−7.3	4.6	28	43.7
16	−8	5	27.1	38.6	41	−16	7.8	30.2	47.5
17	−16	10	30	47.1	42	−5	6.9	25.3	30
18	−5	10	23.7	30	43	−9	5	18.4	40
19	−9	5	22.8	41.5	44	−18	10	35.9	40.5
20	−18	15	35.3	43.7	45	−8	7.3	25	37.9
21	−8	10	25.3	53.1	46	−7.1	7.7	31.2	49
22	−8	5	30.47	47.3	47	−16	5.8	30	37.9
23	−16	5	30.1	40.1	48	−5	6	25	30
24	−5	6	25.3	37.9	49	−15	8.1	20	40
25	−9	5	10	45.8	50	−10	5	14.7	37.9

Table 2.1: Coefficients for the functions f_i in (2.38)

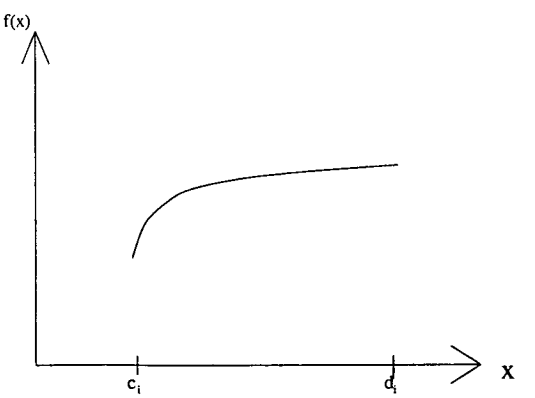

Figure 2.1: shape of f_i functions

In the notation of section 2.4, $D_i = [0, d_i] \quad \forall i$, so $\partial D_i = \{0, d_i\}$. Each f_i is differentiable except at the point c_i. For optimal $\widehat{\boldsymbol{x}}$ the necessary optimality conditions from (2.13) are:

$$\exists(\lambda_0, \lambda) \text{ with } \lambda_0 \in \{0,1\}, \quad \lambda \in I\!R, \quad (\lambda_0, \lambda) \neq (0,0) \text{ such that for every } i:$$
$$\lambda_0 \frac{\mathrm{d}f_i}{\mathrm{d}x}(\widehat{x}_i) = \lambda \qquad \text{if } \widehat{x}_i \neq 0, \widehat{x}_i \neq c_i, \widehat{x}_i \neq d_i.$$

If $\lambda_0 = 0$ in the above condition then it implies $\lambda = 0$, but this contradicts $(\lambda_0, \lambda) \neq (0,0)$. Hence $\lambda_0 = 1$. Therefore the necessary condition simplifies to:

$$\exists\lambda \text{ such that for every } i: \qquad \frac{\mathrm{d}f_i}{\mathrm{d}x}(\widehat{x}_i) = \lambda \qquad \text{if } \widehat{x}_i \neq 0, \widehat{x}_i \neq c_i, \widehat{x}_i \neq d_i.$$

Graphically, this condition simply says that at the optimal solution all the functions f_i, for which the argument is in the interior of the feasible set and is a point at which f_i is differentiable, have the same gradient. Note that since the f_i functions are built of log-functions, there is a quick way (in this case even analytic) to identify those $x_i \in I\!R$ for which $\frac{\mathrm{d}f_i}{\mathrm{d}x}(x_i) = \lambda$ when λ is known. Hence the FJ action elimination will be very effective.

For dynamic programming we introduce a discretisation of the x-axis. The interval $[0, \hat{q}]$ is split into n equal subintervals, of length δ.

$$\delta = \frac{\hat{q}}{n} = \frac{1000}{n}$$

Let n be called the discretisation number. Only x_i values of the form $x_i = k\delta$ with $k \in I\!N$ are considered.

For the continuous version of the problem discussed earlier the states, value function and actions are $(q, i), F_i(q), A_i(q)$ respectively, where $q \in I\!R$ and $i \in I\!N$. For the discrete version of the problem the notation $(j, i), \overline{F}_i(j), \overline{A}_i(j)$ will be used for states, value function and optimal actions respectively.

Definition 2.6.1

$(\mathbf{j}, \mathbf{i})$ is the state considering the first i functions f_h only, $\sum_{l=1}^{i} x_l = j\delta$ and each x_l is either zero or a positive multiple of δ.

$\overline{\mathbf{F}}_\mathbf{i}(\mathbf{j})$ = maximal value of $\sum_{l=1}^{i} f_l(x_l)$ where $\boldsymbol{x} \in I\!R^i$ and belongs to state (j, i).

$\overline{\mathbf{A}}_\mathbf{i}(\mathbf{j})$ = computed number of units (δ) that x_i takes, when $\boldsymbol{x} \in I\!R^i$ and $\boldsymbol{x}$ belongs to state (j, i) and $\sum_{l=1}^{i} f_l(x_l) = \overline{F}_i(j)$

Some remarks on these definitions: (j, i) can be thought of as a finite set of vectors $\boldsymbol{x} \in I\!R^i$. $\overline{F}_i(j)$ is the solution of a maximization problem with finite search space. $\overline{A}_i(j)$ is a discrete action.

As observed before the example in this section is such that (2.13) implies that the FJ-conditions always hold with $\lambda_0 = 1$. Furthermore for this example $B_i(q)$ takes two possible forms: Either $B_i(q) = \{(1, \lambda)\}$ for some $\lambda \in I\!R$ or $B_i(q) = \{(1, \lambda) : \lambda \in I\!R\}$. However, when $B_i(q) = \{(1, \lambda) : \lambda \in I\!R\}$, the action elimination step will actually not eliminate any actions that ordinary DP considers. Hence in this case the action elimination step will not be used. For simplification a function λ of the states will be used instead of the B-sets in the description of the following algorithm. For any state, if $B_i(q) = \{(1, \lambda)\}$ for some $\lambda \in I\!R$ then $\lambda(q, i)$ will be defined to be this unique λ-value, and if $B_i(q) = \{(1, \lambda) : \lambda \in I\!R\}$ then $\lambda(q, i)$ will be said to be undefined. Only if $\lambda(q, i-1)$ is defined will the FJ-action elimination be used. Step 3 of the algorithm below is based on the procedure in section 2.4.1

Algorithm 2.6.2

1) Choose $n \in I\!N$ and set $\delta := \dfrac{\hat{q}}{n}$

$S_1 := \{$ all special points of f_1 in $[0, d_1]\} = \{0, c_1, d_1\}$

$Q := \left\{ \lfloor \frac{y}{\delta} \rfloor, \lceil \frac{y}{\delta} \rceil \quad : y \in S_1 \right\}$

for $j := 0$ to n do

if $j\delta \leq d_1$ then

$\overline{F}_1(j) := f_1(j\delta)$

$\overline{A}_1(j) := j$

if $j \notin Q$ then

$\lambda(j, 1) := \dfrac{\mathrm{d}f_1}{\mathrm{d}x}(j\delta)$

else

$\lambda(j, 1) := undefined$

end if

else

$\overline{F}_1(j) := -\infty$

end if

end do j

2) $i := 2$

3) for $j := 0$ to n set $\overline{F}_i(j) := -\infty$

$S_i := \{$ all special points of f_i in $[0, d_i]\} = \{0, c_i, d_i\}$

$Q := \left\{ \lfloor \frac{y}{\delta} \rfloor, \lceil \frac{y}{\delta} \rceil \quad : y \in S_i \right\}$

for $j \in \{0, 1, ..., n\}$ for which $F_{i-1}(j) > -\infty$ do

if $\lambda(j, i-1)$ is defined then

$$J := \left\{ x \in I\!R : 0 < x \leq \min(d_i, \hat{q} - j\delta), \frac{\mathrm{d}f_i}{\mathrm{d}x}(x) = \lambda(j, i-1) \right\}$$

for all $k \in \left\{ \lfloor \frac{y}{\delta} \rfloor, \lceil \frac{y}{\delta} \rceil \quad : y \in J \right\} \cup Q$ do

if $\overline{F}_i(j+k) < \overline{F}_{i-1}(j) + f_i(k\delta)$ then

$\overline{F}_i(j+k) := \overline{F}_{i-1}(j) + f_i(k\delta)$

$\overline{A}_i(j+k) := k$

$\lambda(j+k, i) := \lambda(j, i-1)$

end if

end do k

else (if $\lambda(j, i-1)$ is not defined)

for $k := 0$ to $\min(n-j, \lfloor \frac{d_i}{\delta} \rfloor)$ do

if $\overline{F}_i(j+k) < \overline{F}_{i-1}(j) + f_i(k\delta)$ then

$\overline{F}_i(j+k) := \overline{F}_{i-1}(j) + f_i(k\delta)$

$\overline{A}_i(j+k) := k$

if $k \notin Q$ then

$\lambda(j+k, i) := \frac{\mathrm{d}f_i}{\mathrm{d}x}(k\delta)$

else (if $k \in Q$)

$\lambda(j+k, i) := undefined$

end if

end if

end do k

end if

end do j

4) if $i < r$ increase i by one and return to 3)

5) $s := n$

for $i := r$ down to 1 do

$x_i^* := \overline{A}_i(s)\delta$

$s := s - \overline{A}_i(s)$

end do i

stop

Step 1 is the initialisation of DP, step 3 is based on the procedure of section 2.4.1

n	ordinary DP (secs)	FJ-DP (secs)	$\overline{F}_r(n)$
100	0.04	0.06	432.81
200	0.16	0.13	447.33
500	0.87	0.35	456.79
1000	3.14	0.68	458.23
2000	12.41	1.52	459.35
10000	326.43	7.64	460.12

Table 2.2: computational results

and calculates the value function $\overline{F}_i(j)$ and the optimal action $\overline{A}_i(j)$ for the states in stage i, step 5 calculates the optimal distribution vector $\boldsymbol{x}^*$. $\overline{F}_r(n)$ is accepted as the optimal value of the objective. Observe that $\boldsymbol{x}^*$ is restricted to the grid, which implies that all components of $\boldsymbol{x}^*$ are an integer multiple of δ.

The above FJ-DP algorithm was used to solve problem (2.37) for various discretisation numbers n. The results are compared with those of DP without action elimination and are given in Table 2.2. For the same discretisation number n the ordinary DP and the FJ-DP algorithms gave the same optimum $\overline{F}_r(n)$. Since it takes $O(1)$ amount of work to identify those $x_i \in I\!R$ for which $\frac{\mathrm{d}f_i}{\mathrm{d}x}(x_i) = \lambda$ when λ is known, we expect the complexity of the FJ-DP for this problem to be $O(n)$. The complexity of ordinary DP is $O(n^2)$. Our run time results are consistent with this analysis.

This example has a similar form to the gaslift allocation problem with gaslift availability constraint only, described in the introduction. The main difference is that in a gaslift problem the functions f_i are normally not logarithmic functions given analytically but functions given numerically as a set of points.

2.6.1 Finding points with a particular derivative of f_i

For the FJ-action elimination in Algorithm 2.6.2 there must be a procedure which gives all points x_i for a given value of λ such that

$$\frac{\mathrm{d}f_i}{\mathrm{d}x}(x_i) = \lambda. \tag{2.39}$$

In the last example this was done analytically since the functions f_i were of a suitable analytic form. However, if the solution of (2.39) can not be done analytically this can be done fast numerically by preprocessing the function f_i before doing the DP-calculation of going from stage $i-1$ to stage i. For example, if the functions f_i were arbitrary piecewise smooth functions the preprocessing of the functions f_i could be done as follows: the domain of x_i is split into subsets, such that on each subset the function f_i is either convex or concave. Then for

each subset a function $x(\lambda)$ is approximated where $x(\lambda)$ satisfies

$$\frac{\mathrm{d}f_i}{\mathrm{d}x}(x(\lambda)) = \lambda$$

$x(\lambda)$ can be calculated for a list of increasing or decreasing λ-values efficiently using Newton's method or binary bisection. Alternatively, (2.39) can be solved approximately to within ϵ of the true solution by simply applying binary bisection to every convex or concave subset of the domain of x_i. This method with $\epsilon = \frac{\delta}{2}$ was implemented for the previous example problem and the results for different discretisation numbers n are given below.

n	FJ − DP , binary bisection (secs)	$\overline{F}_r(n)$
100	0.07	432.81
200	0.29	447.33
500	0.53	456.79
1000	0.91	458.23
2000	1.80	459.35
10000	9.98	460.12

When comparing this table with Table 2.2 it is found that the result $\overline{F}_r(n)$ is as before for the considered discretisation numbers n. This is not surprising. When (2.39) with $\lambda > 0$ is solved by binary bisection to within $\epsilon = \frac{\delta}{2}$ of the true solution the work necessary is $O(\log n)$. Hence the complexity of the FJ-DP algorithm now is expected to be $O(n \log n)$. The results of the table are consistent with this analysis.

2.6.2 A modified example

Algorithm 2.6.2 uses a regular discretisation with step size δ for the action space and for the state space in each stage. Furthermore, the problem solved has one linear constraint with all coefficients being equal to one. This has the consequence that when the constraint function is evaluated at any discretisation point $\boldsymbol{x}$ the result is a value which is a discretisation point of the state space. It is important to realise that this property is not true in general. In this subsection an example is given which makes this point clear. In general it is not enough to have some discretisation points in the state space but it is necessary to subdivide the state space into subintervals. Consider this problem:

$$\begin{aligned} \text{maximize} f(\boldsymbol{x}) := & \sum_{h=1}^{r} f_h(x_h) \\ \text{subject to} \quad & \sum_{h=1}^{r} w_h(x_h) = \hat{q}, \\ x_h \in I\!R, \quad & x_h \in [0, d_h] \qquad 1 \le h \le r, \end{aligned} \tag{2.40}$$

where the functions f_h, w_h are $\mathbb{R} \to \mathbb{R}$. The functions f_h, w_h are everywhere differentiable on $[0, d_h]$ except at finitely many points. Let's also assume that $w_h(x_h) \geq 0$ and that $w_h'(x_h) \neq 0$ for all $x_h \in [0, d_h]$ and that $\hat{q} > 0$. These assumptions simplify the description of Algorithm A.0.1. Let N_h be the set of points at which either f_h or w_h is not differentiable. For optimal $\widehat{\boldsymbol{x}}$ the necessary optimality conditions from (2.13) are:

$$\exists(\lambda_0, \lambda) \text{ with } \lambda_0 \in \{0,1\}, \quad \lambda \in \mathbb{R}, \quad (\lambda_0, \lambda) \neq (0,0) \text{ such that for every } i:$$
$$\lambda_0 f_i'(\widehat{x}_i) = \lambda w_i'(\widehat{x}_i) \quad \text{if } \widehat{x}_i \neq 0, \widehat{x}_i \neq d_i, \widehat{x}_i \notin N_i.$$

If $\lambda_0 = 0$ in the above condition then it implies $\lambda = 0$ since $w_i'(x_i) \neq 0$ for all $x_i \in [0, d_i]$ by assumption, but this contradicts $(\lambda_0, \lambda) \neq (0,0)$. Hence $\lambda_0 = 1$. Therefore the necessary condition simplifies to:

$$\exists\lambda \text{ such that for every } i: \quad f_i'(\widehat{x}_i) = \lambda w_i'(\widehat{x}_i) \text{ if } \widehat{x}_i \neq 0, \widehat{x}_i \neq d_i, \widehat{x}_i \notin N_i.$$

It is important to be aware of the differences between the definitions below and those of Definition 2.6.1. δ is again defined by $\delta := \frac{\hat{q}}{n}$ where n is the discretisation number.

Definition 2.6.3

$\mathbf{(j,i)}$ is the state considering the first i functions f_h only,
$(j-1)\delta < \sum_{h=1}^{i} w_h(x_h) \leq j\delta$ and each $x_h \in D_h$ for $1 \leq h \leq i$.

$\mathbf{F_i(j)}$ =maximal value of $\sum_{h=1}^{i} f_h(x_h)$ where $\boldsymbol{x} \in \mathbb{R}^i$ belongs to state (j,i).

$\overline{\mathbf{F}}_\mathbf{i}\mathbf{(j)}$ =computed approximation to $F_i(j)$.

$\overline{\mathbf{A}}_\mathbf{i}\mathbf{(j)}$ =computed real value that x_i takes, when $\boldsymbol{x} \in \mathbb{R}^i$ and $\boldsymbol{x}$ belongs to state (j,i) and $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j)$

$\mathbf{p_i(j)}$ $=\sum_{h=1}^{i} w_h(x_h)$ where $\boldsymbol{x} \in \mathbb{R}^i$ is the computed vector which belongs to state (j,i) and for which $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j)$.

Some remarks on these definitions: (j,i) can be thought of as an infinite set of vectors $\boldsymbol{x} \in \mathbb{R}^i$. $F_i(j)$ is the solution of a maximization problem with infinite search space. DP does finite computation, it can not compute $F_i(j)$ exactly in general. DP will approximate $F_i(j)$, the approximation is denoted by $\overline{F}_i(j)$. $\overline{A}_i(j)$ is a real valued action. By definition $p_i(j)$ satisfies:

$$(j-1)\delta < p_i(j) \leq j\delta.$$

Suppose $\boldsymbol{x} \in I\!R^i$ is the computed vector which belongs to state (j,i) and for which $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j)$. In the DP process $i+1$ dimensional vectors of the form $(\boldsymbol{x}, x_{i+1})$ will be considered and using $p_i(j)$ such $i+1$ dimensional vectors can be assigned to the right state of stage $i+1$.

The algorithm in Appendix A is an implementation of FJ-DP for problem (2.40), its structure is similar to the previous algorithm (Algorithm 2.6.2) and in particular step 3 is again based on the procedure in section 2.4.1.

If the discretisation number n is sufficiently large and if problem (2.40) is infeasible then the algorithm will return $F_r(n) = -\infty$. Notice that in general the result of this algorithm $\boldsymbol{x}^*$ does not satisfy the equality constraint exactly but contains a discretisation error with the following bound:

$$\hat{q} - \delta < \sum_{h=1}^{r} w_h(x_h^*) \leq \hat{q}$$

2.6.3 Remarks on storage issues

In the FJ-DP algorithm of the previous section (the algorithm is explicitly given in Appendix A) the following objects are used:
$\overline{F}_i(j)$, $\overline{A}_i(j)$, $p_i(j)$ and $\lambda(j,i)$ for $1 \leq i \leq r, \quad 0 \leq j \leq n$. However, it is not necessary to store all of these objects. When step 3 of the algorithm is executed the only objects used there are $\overline{F}_{i-1}(j)$, $\overline{F}_i(j)$, $\overline{A}_i(j)$, $p_{i-1}(j)$, $p_i(j)$, $\lambda(j,i-1)$, $\lambda(j,i)$ for $0 \leq j \leq n$. The objects $\overline{A}_t(j)$ for $t \leq i$, $0 \leq j \leq n$ will be needed in step 5, but $\overline{F}_t(j)$, $p_t(j)$, $\lambda(j,t)$ for $t < i-1$, $0 \leq j \leq n$ are no longer needed. Hence the algorithm only needs the following storage:
$2(n+1)$ values (instead of $r(n+1)$ values) for each of the objects $\overline{F}$, p, λ and $r(n+1)$ values for $\overline{A}$.

2.6.4 Calculating numerically the action set when state and action space are both one dimensional

This subsection is a generalisation of the first part of section 2.6.1. The general deterministic DP problem with one dimensional state and action space in each stage is:

$$\begin{aligned} \text{maximize} \quad & f(\boldsymbol{x}) := \sum_{h=1}^{r} f_h(x_h) \\ \text{subject to} \quad & \sum_{h=1}^{r} w_h(x_h) = \hat{q}, \\ & x_h \in I\!R, \quad x_h \in D_h \qquad 1 \leq h \leq r, \end{aligned}$$

where the functions f_h, w_h are piecewise differentiable functions $\mathbb{R} \to \mathbb{R}$ with finitely many non-differentiable points. Let N_h be the set of points $x \in D_h$ at which either f_h or w_h is not differentiable. Suppose that this problem is solved by DP using some discretisation scheme where the state space and action space are discretised in n units. DP without action elimination needs $O(n^2)$ operations to do the calculation of going from stage $i-1$ to stage i for $i \geq 2$. FJ-DP, when implemented with a procedure from section 2.4.1, has to calculate about n times sets J of the form

$$J := \{x : x \in D_i, \lambda_0 f_i'(x) = \lambda w_i'(x)\}$$

for given (λ_0, λ). Often the functions f_i, w_i do not allow the points x of J to be calculated analytically. In the sequel it is shown how this can be done numerically. For this it is assumed that the functions f_i, w_i are piecewise continuously differentiable not just piecewise differentiable. The two cases $\lambda_0 = 0$ and $\lambda_0 = 1$ have to be considered separately. Let's first consider $\lambda_0 = 0$: in this case $\lambda \neq 0$ since the FJ conditions demand $(\lambda_0, \lambda) \neq (0, 0)$. This means the calculation of J reduces to that of J_0 where

$$J_0 := \{x : x \in D_i, w_i'(x) = 0\}$$

Pick a small number $\epsilon > 0$ where ϵ must satisfy $|y - z| > 2\epsilon$ for all $y, z \in N_i$ and $y \neq z$. Define

$$X_+ := \{x : x \in D_i, x = y + \epsilon, y \in N_i\}$$
$$\text{and} \quad X_- := \{x : x \in D_i, x = y - \epsilon, y \in N_i\}.$$

The elements of J_0 can be calculated approximately by running once through the discretised actions $x \in D_i$ and through the points of X_+ and X_- and looking at the corresponding values of w_i'. If $a < b$, a is a discretisation point or $a \in X_+$, b is a discretisation point or $b \in X_-$, a and b are close and either

$$w_i'(a) < 0 \quad \text{and } w_i'(b) > 0,$$
$$\text{or} \quad w_i'(a) > 0 \quad \text{and } w_i'(b) < 0$$

then a fixed number of binary bisection steps or a few iterations of Newton's Method will approximately find a point $x \in J_0$ which lies between a and b. The work involved to find the points of J_0 approximately this way is $O(n)$.

For the case $\lambda_0 = 1$ it is good to get some information about the behaviour of $\frac{f_i'(x)}{w_i'(x)}$ before doing the DP step of going from stage $i-1$ to stage i. By running once

through the discretisation points of D_i, X_+ and X_- non-overlapping maximal intervals can be determined such that on each of these intervals f_i and w_i are differentiable and $\frac{f_i'(x)}{w_i'(x)}$ is either increasing or decreasing. After that, when doing the DP step of going from stage $i-1$ to stage i, on every interval the solutions of

$$\frac{f_i'(x)}{w_i'(x)} = \lambda$$

can be found by the bisection method. This way the elements of J, for any given λ, can approximately be found with $O(\log n)$ operations. Hence, it can be expected that the work of going from stage $i-1$ to stage i of FJ-DP takes $O(n \log n)$ operations.

2.7 Example with two constraints

In this section an example is given with $n_i = 1$ and $m = 2$.

Consider the problem:

$$\begin{aligned} \text{maximize} f(\boldsymbol{x}) := & \sum_{h=1}^{r} \frac{h}{2h+1} f_h(x_h) \\ \text{subject to} \quad & \sum_{h=1}^{r} \frac{h+1}{2h+1} f_h(x_h) = \hat{q}_1, \\ & \sum_{h=1}^{r} x_h = \hat{q}_2, \\ x_h \in I\!R, \quad & x_h \in [0, d_h] \qquad 1 \le h \le r, \end{aligned} \tag{2.41}$$

where $r = 15$, $\hat{q}_1 = 60$ and $\hat{q}_2 = 150$. Let the functions f_i be those given in (2.38) with the numbers a_i, b_i, c_i, d_i taken again from Table 2.1. The domain of x_i can be taken as $D_i = [0, d_i]$. For optimal $\widehat{\boldsymbol{x}}$ the necessary optimality conditions from (2.13) are:

$$\begin{aligned} &\exists (\lambda_0, \lambda) \text{ with } \lambda_0 \in \{0,1\}, \quad \lambda \in I\!R^2, \quad (\lambda_0, \lambda) \neq (0,0) \text{ such that for every } i: \\ &\lambda_0 \frac{i}{2i+1} \frac{\mathrm{d}f_i}{\mathrm{d}x}(\widehat{x}_i) = \lambda_1 \frac{i+1}{2i+1} \frac{\mathrm{d}f_i}{\mathrm{d}x}(\widehat{x}_i) + \lambda_2 \qquad \text{if } \widehat{x}_i \neq 0, \widehat{x}_i \neq c_i, \widehat{x}_i \neq d_i. \end{aligned}$$

This condition implies the following condition

$$\frac{\mathrm{d}f_i}{\mathrm{d}x}(\widehat{x}_i) = \frac{(2i+1)\lambda_2}{i\lambda_0 - (i+1)\lambda_1} \qquad \text{if } \widehat{x}_i \neq 0, \widehat{x}_i \neq c_i, \widehat{x}_i \neq d_i \quad \text{and } i\lambda_0 - (i+1)\lambda_1 \neq 0.$$

This optimality condition is the basis for the FJ action elimination when DP goes from stage $i-1$ to stage i in this problem with one dimensional action and two dimensional state space. Problem (2.41) has been implemented and the

n	ordinary DP (secs)	FJ-DP (secs)	speed-up	$\overline{F}_r(n)$
20	0.14	0.09	1.56	45.58
50	1.59	0.62	2.56	47.80
100	10.54	2.67	3.95	48.50
200	79.37	12.02	6.60	48.51

Table 2.3: computational results

computational results are shown in Table 2.3. The discretisation used discretises D_i in n regular subintervals and also each of the two dimensions of the state space in n regular subintervals. Exactly the same discretisation scheme was used for ordinary DP and FJ-DP. $\overline{F}_r(n)$ is the computed optimal objective value when the discretisation number is n. For the same discretisation number n the computed optimal objective value of ordinary DP and of FJ-DP were always the same. For the FJ-DP it is crucial to quickly find all $x_i \in D_i$ satisfying

$$\frac{\mathrm{d}f_i}{\mathrm{d}x}(x_i) = \mu \tag{2.42}$$

for any given μ (μ depends on i, λ_0, λ_1, λ_2). In this example equation (2.42) can be solved analytically with $O(1)$ work since the functions f_i are made of logarithmic functions. If (2.42) had to be solved numerically it would require at most $O(\log n)$ work since binary bisection could be used. (By investigating f_i before doing the DP recursion, in practice the numerical solution of (2.42) can still be done with $O(1)$ work.) For the results shown in Table 2.3 (2.42) was solved analytically.

The complexities of ordinary DP and FJ-DP on this problem are as follows: there are $O(n^2)$ states in each stage. Ordinary DP checks $O(n)$ actions from each state when going from stage $i-1$ to stage i. There are r stages. Hence the complexity of ordinary DP is expected to be $O(rn^3)$. For FJ-DP the analysis is trickier. Note that in stage 1 only $O(n)$ states are actually defined with value function not equal to $-\infty$ because the action space is one-dimensional and is mapped into the state space of stage 1. The action elimination can only eliminate actions when $B_i(q)$ has finitely many elements (λ_0, λ). By looking at (2.6) one can see that $B_1(q)$ always has infinitely many elements because (λ_0, λ) is in $B_1(q)$ if (λ_0, λ) satisfies one equation, but $\lambda \in I\!R^2$. For $i \geq 2$ it can be expected that $B_i(q)$ is finite for most states since then $(\lambda_0, \lambda) \in B_i(q)$ if (λ_0, λ) satisfies i equations, *i.e.* (λ_0, λ) satisfies a number of equations which is at least as big as the dimension of λ. Note that since λ_0 is restricted to $\{0, 1\}$ only the dimension of λ is important to determine how many equations are necessary so that the set of (λ_0, λ) satisfying the equations is finite. When going from stage 1 to stage 2

FJ-DP checks $O(n)$ actions from each of $O(n)$ states. Therefore the transition from stage 1 to stage 2 takes $O(n^2)$ operations. When going from stage $i-1$ to stage i for $i \geq 3$ FJ-DP checks $O(1)$ actions from most of the $O(n^2)$ states in stage $i-1$ and checks $O(n)$ actions from some of the states in stage $i-1$. It is reasonable to assume that "some of the states" in the last sentence is of order $O(n)$, *i.e.* one order of magnitude less than the total number of states in stage $i-1$. Therefore, the transition to the next stage can be expected to be $O(n^2)$. There are $r-1$ transitions of stages to be done. Hence the total complexity of FJ-DP on this problem is expected to be $O(rn^2)$. The computational results of Table 2.3 support this estimate.

2.8 Conclusions

An action elimination procedure for deterministic dynamic programming problems based on first order optimality conditions has been presented in this chapter. The class of problems considered were of the form (2.1),(2.2) with some or all equality constraints in (2.2) possibly being inequality constraints instead. The efficiency of the action elimination depends on the interior of the domain D_i being large in comparison to the boundary ∂D_i. Another condition for the action elimination to be efficient is that there is a quick method to identify those $x_i \in D_i$ which satisfy (2.10) for any given (λ_0, λ).

Section 2.2 presented the theoretical basis for the action elimination based on the FJ conditions. Then section 2.3 showed how to practically implement the action elimination when going from stage $i-1$ to stage i in a DP recursion. Section 2.4 showed that it is not necessary that the functions in (2.1),(2.2) are differentiable everywhere in order to apply FJ action elimination. This is important because this extension increases considerably the applicability of FJ action elimination.

Section 2.5 discussed how to handle inequality constraints in an efficient way. Sections 2.6 and 2.7 showed details of implementations of example problems. The computational results support the claim that FJ action elimination reduces the computation time by an order of magnitude.

Chapter 3

Discretisation issues

3.1 Variable grid FJ-DP method

In this section a variable grid discretisation for FJ-DP will be presented. It will be illustrated by applying it to solve problem (2.37). The discretisation step length for $x_i \in D_i$ is ϵ_i and for the argument of the value function F_i it is δ_i. These discretisation step lengths are given by

$$\epsilon_i = \frac{d_i}{n} \quad \text{and} \quad \delta_i = \frac{\min\left(\sum_{h=1}^{i} d_h, \quad \hat{q}\right)}{n}$$

where n is the discretisation number. The motivation for this is to have about n discretisation points for x_i and for the argument of the value function F_i in the relevant region of these objects. This idea is taken from interpolation methods. As will be seen at the end this discretisation for the FJ-DP achieves a much better result $\overline{F}_r(n)$ than the discretisation used in Algorithm 2.6.2, particularly when n is small. There are subtle differences between the definitions of states, actions, value function below and those of Definition 2.6.1. The main differences are that the state space is subdivided into a finite number of subintervals rather than a finite number of discretisation points being picked from the state space, secondly, the discretisation length δ varies with the stages.

Definition 3.1.1

$(\mathbf{j}, \mathbf{i})$ is the state considering the first i functions f_h only,
$(j-1)\delta_i < \sum_{h=1}^{i} x_h \leq j\delta_i$ and each $x_h \in D_h$ for $1 \leq h \leq i$.

$\mathbf{F_i(j)}$ =maximal value of $\sum_{h=1}^{i} f_h(x_h)$ where $\boldsymbol{x} \in I\!R^i$ belongs to state (j, i).

$\overline{\mathbf{F}}_\mathbf{i}\mathbf{(j)}$ =computed approximation to $F_i(j)$.

$\overline{\mathbf{A}}_\mathbf{i}\mathbf{(j)}$ =computed real value that x_i takes, when $\boldsymbol{x} \in I\!R^i$ and $\boldsymbol{x}$ belongs to state (j, i) and $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j)$

$\mathbf{p_i(j)} = \sum_{h=1}^{i} x_h$ where $\boldsymbol{x} \in I\!R^i$ is the computed vector which belongs to state (j,i) and for which $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j)$.

Some remarks on these definitions: (j,i) can be thought of as an infinite set of vectors $\boldsymbol{x} \in I\!R^i$. $F_i(j)$ is the solution of a maximization problem with infinite search space. $F_i(j)$ is approximated by $\overline{F}_i(j)$. $\overline{A}_i(j)$ is a real valued action. By definition $p_i(j)$ satisfies:

$$(j-1)\delta_i < p_i(j) \leq j\delta_i.$$

$p_i(j)$ is introduced for the correct assignment of vectors of the form $(\boldsymbol{x}, x_{i+1})$ to states when $\boldsymbol{x} \in I\!R^i$ is the computed vector which belongs to state (j,i) and for which $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j)$.

The algorithm below is based on Algorithm 2.6.2, but it uses the variable grid.

Algorithm 3.1.2

1) Choose $n \in I\!N$

$\epsilon_1 := \dfrac{d_1}{n} \quad \delta_1 := \dfrac{\min(d_1, \hat{q})}{n} = \epsilon_1$

$S_1 := \{$ all special points of f_1 in $[0, d_1]\} = \{0, c_1, d_1\}$

$Q := \left\{ \lfloor \dfrac{y}{\epsilon_1} \rfloor, \lceil \dfrac{y}{\epsilon_1} \rceil \quad : y \in S_1 \right\}$

for $j := 0$ to n do

$\overline{F}_1(j) := f_1(j\epsilon_1)$

$\overline{A}_1(j) := j\epsilon_1$

$p_1(j) := j\epsilon_1$

if $j \notin Q$ then $\lambda(j,1) := \dfrac{\mathrm{d}f_1}{\mathrm{d}x}(j\epsilon_1)$

else $\lambda(j,1) := undefined$

end do j

2) $i := 2$

3) for $j := 0$ to n set $\overline{F}_i(j) := -\infty$

$S_i := \{$ all special points of f_i in $[0, d_i]\} = \{0, c_i, d_i\}$

$\delta_i := \dfrac{\min(\sum_{t=1}^{i} d_t, \hat{q})}{n}, \quad \epsilon_i := \dfrac{d_i}{n}$

$Q := \left\{ \lfloor \dfrac{y}{\epsilon_i} \rfloor, \lceil \dfrac{y}{\epsilon_i} \rceil \quad : y \in S_i \right\}$

for $j \in \{0, 1, ..., n\}$ for which $F_{i-1}(j) > -\infty$ do

if $\lambda(j, i-1)$ is defined then

$J := \left\{ x \in I\!R : 0 < x \le d_i, \frac{\mathrm{d}f_i}{\mathrm{d}x}(x) = \lambda(j, i-1) \right\}$

for all $x \in J \cup S_i$ do

$k := \lceil \frac{p_{i-1}(j) + x}{\delta_i} \rceil$

if $k \le n$ and $\overline{F}_i(k) < \overline{F}_{i-1}(j) + f_i(x)$ then

$\overline{F}_i(k) := \overline{F}_{i-1}(j) + f_i(x)$

$\overline{A}_i(k) := x$

$p_i(k) := p_{i-1}(j) + x$

$\lambda(k, i) := \lambda(j, i-1)$

end if

end do x

else (if $\lambda(j, i-1)$ is not defined)

for $t := 0$ to n do

$k := \lceil \frac{p_{i-1}(j) + t\epsilon_i}{\delta_i} \rceil$

if $k \le n$ and $\overline{F}_i(k) < \overline{F}_{i-1}(j) + f_i(t\epsilon_i)$ then

$\overline{F}_i(k) := \overline{F}_{i-1}(j) + f_i(t\epsilon_i)$

$\overline{A}_i(k) := t\epsilon_i$

$p_i(k) := p_{i-1}(j) + t\epsilon_i$

if $t \notin Q$ then

$\lambda(k, i) := \frac{\mathrm{d}f_i}{\mathrm{d}x}(t\epsilon_i)$

else (if $t \in Q$)

$\lambda(k, i) := undefined$

end if

end if

end do t

end if

end do j

4) if $i < r$ increase i by one and return to 3)

5) if $F_r(n) > -\infty$ do

$s := n$

$\tilde{p} := p_r(n)$

for $i := r$ down to 1 do

$x_i^* := \overline{A}_i(s)$

$\tilde{p}_{new} := \tilde{p}_{old} - \overline{A}_i(s)$

$s := \left\lceil \frac{\tilde{p}}{\delta_{i-1}} \right\rceil$

end do i

end if

stop

As in Algorithm A.0.1, if the discretisation number n is sufficiently large and if problem (2.37) were infeasible then the algorithm would return $F_r(n) = -\infty$. The result $\boldsymbol{x}^*$ does not satisfy the equality constraint exactly but contains a discretisation error with the following bound:

$$\hat{q} - \delta_r < \sum_{h=1}^{r} x_h^* \leq \hat{q} \tag{3.1}$$

The reason for using the variable grid is to achieve good results $\overline{F}_r(n)$ for relatively small discretisation numbers n. However, smaller n means that δ_r is larger and hence the above bound implies that the violation of the equality constraint by $\boldsymbol{x}^*$ can be larger. This suggests that it is a good idea to do a local optimization with $\boldsymbol{x}^*$ as starting point after the DP when n is small.

3.1.1 Computational results

For the discretisation numbers 20,40,60,80,100,200,500 Algorithm 3.1.2 was run. Equations of the form (2.39) were solved analytically. The results are given in Table 3.1. Comparing the results of Table 3.1 with those of Table 2.2 it is

n	$F_r(n)$	$p_r(n)$ $(= \sum_{h=1}^{r} x_h^*)$	run time (secs)
20	393.57	986.93	0.03
40	433.93	985.22	0.03
60	450.30	999.56	0.05
80	451.80	994.76	0.07
100	453.37	995.35	0.11
200	456.77	998.59	0.24
500	457.08	999.85	0.59
1000	457.06	999.16	1.39
2000	457.24	999.87	2.58
10000	460.03	999.98	14.27

Table 3.1: Results of FJ-DP with variable grid

clear that the variable grid FJ-DP method gives much better results for small discretisation numbers n than does the fixed grid FJ-DP method of section 2.6.

Secondly, for the same discretisation number n the run time of the variable grid FJ-DP is roughly twice the run time of the fixed grid FJ-DP method. As can be seen from the table the computed optimal vector $\boldsymbol{x}^*$ violates the constraint $\sum_{h=1}^{r} x_h = 1000$. The constraint violation satisfies (3.1) where δ_r is always equal to $\frac{\hat{q}}{n} = \frac{1000}{n}$.

For the discretisation numbers $20, 30, 40, ...1000$ (always increasing by ten) Algorithm 3.1.2 with a local optimization added at the end with $\boldsymbol{x}^*$ as starting point was run. The results are shown in Table 3.2. For those discretisation numbers of the form $n = 10k$ with $k \in I\!N$ and $2 \leq k \leq 100$ for which the result is not explicitly shown in this table the computed optimal objective value was always greater than 459.1. The optimal objective value of problem (2.37) is 460.12 (rounded to 2 decimals). Surprisingly, the discretisation numbers 150,430,440 and 970 worked considerably worse than other discretisation numbers of similar magnitude. The reason for the bad performance of these discretisation numbers is not that they are too small but that in the DP process some optimal actions are eliminated. The next example will deal with this issue.

n	optimal objective value
20	401.72
30	428.49
40	440.28
50	454.58
60	457.26
70	456.75
80	456.20
90	456.11
100	457.10
110	456.22

n	optimal objective value
120	459.12
130	460.12
140	459.21
150	457.10
...	...
430	457.45
440	456.12
...	...
970	456.12
...	...

Table 3.2: Results of FJ-DP with variable grid and local optimization added

3.1.2 Problems with the basic variable grid FJ-DP method

In this section a small example is used to illustrate the type of problem with Algorithm 3.1.2 which led to the bad performance of the discretisation numbers 150,430,440 and 970. At the heart of the problem is that optimal actions are eliminated which can happen if for some state (j, i) the dual multiplier $\lambda(j, i)$ is defined when its value is actually not acceptable as a shadow price for the value function $\overline{F}_i$. This often happens in a stage i when $\epsilon_i << \delta_i$. Consider the

problem:

$$\begin{array}{ll} \text{maximize} & f(x_1,x_2,x_3) := f_1(x_1)+f_2(x_2)+f_3(x_3) \\ \text{subject to} & x_1+x_2+x_3=6, \\ & x_1\in[0,3], x_2\in[0,3], x_3\in[0,3] \end{array} \tag{3.2}$$

where the functions f_1, f_2, f_3 are given by

$$\begin{aligned} f_1(x) &= \begin{cases} 0 & \text{if } 0\le x<1, \\ 0.9+0.1x & \text{if } 1\le x\le 3 \end{cases} \\ f_2(x) &= \begin{cases} 0 & \text{if } 0\le x<2, \\ 1 & \text{if } 2\le x\le 3 \end{cases} \\ f_3(x) &= \sqrt{x+0.1} \end{aligned}$$

By sketching these functions and by inspection it can be seen that the optimal point $\tilde{\boldsymbol{x}}$ of this problem has components $\tilde{x}_1=1$, $\tilde{x}_2=2$, $\tilde{x}_3=3$ which gives 3.76 as objective value (to two decimals rounded). Let a variable grid algorithm similar to Algorithm 3.1.2 be applied to this problem and let the discretisation number be $n=3$. The values for ϵ_i, δ_i will be

$$\begin{array}{lll} \epsilon_1=1, & \epsilon_2=1, & \epsilon_3=1 \\ \delta_1=1, & \delta_2=2, & \delta_3=2 \end{array}$$

Step 1 (the initialisation step) determines $\overline{F}_1(j)$, $\overline{A}_1(j)$, $p_1(j)$ and $\lambda(j,1)$ for $j=0,1,2,3$ as follows:

state $(j,1)$	$\overline{F}_1(j)$	$\overline{A}_1(j)$	$p_1(j)$	$\lambda(j,1)$
(0,1)	0	0	0	undefined
(1,1)	1	1	1	undefined
(2,1)	1.1	2	2	0.1
(3,1)	1.2	3	3	undefined

Then step 3 calculates $\overline{F}_2(j)$, $\overline{A}_2(j)$, $p_2(j)$ and $\lambda(j,2)$ for $j=0,1,2,3$ as:

state $(j,2)$	$\overline{F}_2(j)$	$\overline{A}_2(j)$	$p_2(j)$	$\lambda(j,2)$
(0,2)	0	0	0	undefined
(1,2)	1.1	0	2	0.1
(2,2)	2.1	2	4	0.1
(3,2)	2.2	2	5	undefined

And for the states of stage 3 it calculates:

state $(j,3)$	$\overline{F}_3(j)$	$\overline{A}_3(j)$	$p_3(j)$	$\lambda(j,3)$
(0,3)	0.32	0	0	undefined
(1,3)	1.45	2	2	0.35
(2,3)	2.42	0	4	0.1
(3,3)	3.25	1	6	0.35

From the actions $\overline{A}_i(j)$ and values of $p_i(j)$ step 5 calculates the optimal allocation vector $\boldsymbol{x}^*$ as

$$(x_1^*, x_2^*, x_2^*) = (3, 2, 1)$$

which gives 3.25 (to two decimals) as objective value. The calculated approximation to the optimal point $\boldsymbol{x}^*$ is quite far away from the optimal point (1,2,3). Let's try to find out why the algorithm failed to find the point (1,2,3) or why it didn't find a point which is closer to the optimal point.

Suppose the algorithm had found the point (1,2,3). The optimal path of point (1,2,3) would have to pass through the states (1,1), (2,2) and (3,3). For these states $\overline{F}_i(j)$, $\overline{A}_i(j)$, $p_i(j)$ and $\lambda(j,i)$ should be:

state (j,i)	$\overline{F}_i(j)$	$\overline{A}_i(j)$	$p_i(j)$	$\lambda(j,i)$
(1,1)	1	1	1	undefined
(2,2)	2	2	3	undefined
(3,3)	3.76	3	6	undefined

Looking back at the previous tables the following is observed: For state (1,1) the algorithm calculated $\overline{F}_i(j)$, $\overline{A}_i(j)$, $p_i(j)$ and $\lambda(j,i)$ as desired but for states (2,2) and (3,3) it did not. Both vectors $(x_1, x_2) = (1, 2)$ and $(x_1, x_2) = (2, 2)$ belong to state (2,2), both these vectors were checked in the DP process and the second vector has a slightly higher resulting objective value and hence defines $\overline{F}_2(2)$, $\overline{A}_2(2)$, $p_2(2)$ and $\lambda(2,2)$. Departing from state (2,2) all actions were eliminated except $x_3 = 0$ and $x_3 = 3$ because $\lambda(2,2) = 0.1$ and the function f_3 has no discontinuous points and no point in the interval (0,3) with gradient 0.1. The action $x_3 = 3$ from state (2,2) would lead to state (4,3) which is outside the considered range. If the algorithm had left $\lambda(2,2)$ undefined instead of setting $\lambda(2,2) = 0.1$ then from state (2,2) the actions $x_3 = 0$, $x_3 = 1$, $x_3 = 2$ and $x_3 = 3$ would have been checked and the algorithm would have found the point (2,2,2) as result which is closer to the optimal point (1,2,3) than (3,2,1) is. The point (2,2,2) has objective value 3.55. Observe that $\lambda(2,2) = 0.1$ is not acceptable as shadow price since

$$\overline{F}_2(1) = 1.1 \not\approx \overline{F}_2(2) + \lambda(2,2)(p_2(1) - p_2(2)) = 1.9$$

This is the case because the second components of the two vectors corresponding to $\overline{F}_2(1)$ and to $\overline{F}_2(2)$ lie on different sides of the discontinuous point $x_2 = 2$.

The variable grid FJ-DP algorithm can overcome this problem in the following way: Lagrange multipliers also have an interpretation as shadow prices. One can impose that $\lambda(j,i)$, when defined, must be acceptable as shadow price.

Property 3.1.3 *When in Algorithm 3.1.2* $\lambda(j,i)$ *is defined then*

a) $\overline{F}_i(k) > -\infty$

b) $|\overline{F}_i(k) - \overline{F}_i(j) - \lambda(j,i)(p_i(k) - p_i(j))| \leq \gamma|\lambda(j,i)(p_i(k) - p_i(j))|$

must hold for $k = j-1$ *if* $j > 0$ *and for* $k = j+1$ *if* $j < n$. $\gamma > 0$ *is a fixed tolerance.*

Usually a value for γ between one and two is suitable. Algorithm 3.1.2 can be improved by imposing Property 3.1.3 with, for example, $\gamma = 1.5$ in the following way:

Algorithm 3.1.4

1) same as step 1 of Algorithm 3.1.2
2) same as step 2 of Algorithm 3.1.2
3) same as step 3 of Algorithm 3.1.2
4) for $j := 0$ to n do
   if $\lambda(j,i)$ is defined and Property 3.1.3 does not hold then
   $\lambda(j,i) := undefined$
   end if
   end do j
5) same as step 4 of Algorithm 3.1.2
6) same as step 5 of Algorithm 3.1.2

3.1.3 DP with variable grid but no action elimination

A DP method with variable grid but no FJ action elimination does not work so well as with action elimination. This is because in some sense ordinary DP is too greedy. In general it is good to have $\overline{F}_i(j), p_i(j)$ associated with state (j,i) such that these two values **together** make state (j,i) an attractive state to depart from and take some action when doing the next DP-recursion from state i to stage $i+1$. When doing the DP calculation of going from stage $i-1$ to stage i

it is not good to maximize $\overline{F}_i(j)$ irrespective of optimality conditions. Ordinary DP would try to make $\overline{F}_i(j)$ as large as possible **even** on the expense that $\boldsymbol{x}$ does not satisfy the FJ conditions where $\boldsymbol{x} \in I\!R^i$ is the computed vector belonging to state (j,i) with $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j)$. Ordinary DP would tend to not 'waste resources' *i.e.* to make $p_i(j)$, in the case of problem (2.37), close to $j\delta_i$ in the attempt to increase $\overline{F}_i(j)$. By doing so the computed optimal vector $\boldsymbol{x} \in I\!R^i$ for the subproblem corresponding to state (j,i) would violate the FJ optimality conditions. Furthermore, this move away from satisfying the FJ conditions can be quite big since ϵ_i and δ_i have different scales in general. But once the FJ property is destroyed for a state in stage i it is destroyed for all states in later stages whose optimal path passes through this particular state in stage i. Such local moves away from satisfying the FJ conditions would, of course, add up. At the end, with ordinary DP and a variable grid, it is likely that the computed optimal distribution is far away from satisfying the FJ optimality conditions.

3.1.4 A further improvement

In this subsection a further improvement for Algorithm 3.1.2 is discussed. Algorithm 3.1.2 solves problem (2.37), *i.e.* the objects below refer to problem (2.37).

Suppose $\boldsymbol{x}^1, \boldsymbol{x}^2 \in I\!R^i$ (for some i),

$$\begin{aligned} \tilde{F}_1 &= \sum_{h=1}^{i} f_h(x_h^1), \qquad \tilde{p}_1 = \sum_{h=1}^{i} x_h^1 \\ \tilde{F}_2 &= \sum_{h=1}^{i} f_h(x_h^2), \qquad \tilde{p}_2 = \sum_{h=1}^{i} x_h^2 \end{aligned}$$

and $\tilde{\lambda}_1$ and $\tilde{\lambda}_2$ are the dual variables corresponding to $\boldsymbol{x}^1, \boldsymbol{x}^2$ respectively. Furthermore, $\boldsymbol{x}^1$, $\boldsymbol{x}^2$ both belong to the same state (j,i), *i.e.* $\tilde{p}_1, \tilde{p}_2$ satisfy

$$\begin{aligned} (j-1)\delta_i < \tilde{p}_1 \leq j\delta_i \\ (j-1)\delta_i < \tilde{p}_2 \leq j\delta_i \end{aligned}$$

If $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ are possible vectors to determine $\overline{F}_i(j)$, $p_i(j)$, $\lambda(j,i)$ which one shall be preferred? Algorithm 3.1.2 prefers $\boldsymbol{x}^1$ to $\boldsymbol{x}^2$ if

$$\tilde{F}_1 > \tilde{F}_2 \tag{3.3}$$

If $\tilde{F}_1$ is just a very little bigger than $\tilde{F}_2$ but $\tilde{\lambda}_1$ is much less than $\tilde{\lambda}_2$ and $\tilde{p}_1 > \tilde{p}_2$ then in fact $\tilde{F}_2$ should be preferred to $\tilde{F}_1$ since it makes state (j,i) a better starting state for the determination process of states of stage $i+1$. This is because the shadow price λ is higher for $\boldsymbol{x}^2$. A better rule which also uses the information

contained in $\tilde{\lambda}_1$, $\tilde{\lambda}_2$, $\tilde{p}_1$, $\tilde{p}_2$ is the following:
prefer $\boldsymbol{x}^1$ to $\boldsymbol{x}^2$ if

$$\tilde{F}_1 > \tilde{F}_2 + \lambda(\tilde{p}_1 - \tilde{p}_2) \tag{3.4}$$

$$\text{where} \quad \lambda = \begin{cases} \tilde{\lambda}_1 & \text{if } \tilde{p}_1 < \tilde{p}_2 \\ \tilde{\lambda}_2 & \text{if } \tilde{p}_1 \geq \tilde{p}_2 \end{cases}$$

And if λ is not defined then prefer $\boldsymbol{x}^1$ to $\boldsymbol{x}^2$ if $\tilde{F}_1 > \tilde{F}_2$. Decision rule (3.4) performs better than rule (3.3).

3.1.5 Computational results

An algorithm like Algorithm 3.1.4 which uses decision rule (3.4) instead of (3.3) was run with a local optimization added at the end. The results are shown in Table 3.3. Discretisation numbers of the form $n = 10k$ with $k \in \mathbb{N}$, $22 \leq k \leq 100$ (which are not shown in the table) always had as result an objective value greater than 459.5 and moreover only for six such discretisation numbers (all six less than 500) was the result not equal to 460.12, which is the optimum of problem (2.37). *i.e.* This algorithm seems to be robust with respect to increasing discretisation numbers.

n	optimal objective value	n	optimal objective value
20	438.06	120	459.14
30	453.60	130	459.22
40	451.07	140	459.22
50	455.65	150	459.22
60	453.41	160	459.12
70	457.10	170	459.14
80	459.71	180	460.12
90	459.22	190	459.63
100	459.14	200	459.14
110	458.81	210	460.12

Table 3.3: Results of Algorithm 3.1.4 with rule (3.4) and local optimization added

3.2 FJ action elimination applied to interpolation methods

In this section the FJ action elimination is discussed when it is applied to an interpolation method. The problem considered in this section is problem (2.1), (2.2) with the additional constraint (2.19). Suppose this problem is solved with

a DP interpolation method which uses a discretisation of the state space and action space like the one of section 4.1 (described there in the first paragraph) and the discretisation is regular in each component of q in the state space (q, i) of stage i. The vector δ_i, which is a m-dimensional vector of positive real values, characterises the grid of discretisation points in the state space of stage i. In other words: The discrete state $(j_1, j_2, ..., j_m, i)$ corresponds to the real state (q, i) where the k-th component of q is

$$q_k = j_k \delta_{i,k}.$$

An interpolation method approximates $F_i(q)$ only for discretised states (q, i). Let $\overline{F}_i(q)$ be the calculated approximation to $F_i(q)$ for discretised states (q, i). Also actions $A_i(q)$ are calculated only for discretised states (q, i). When going from stage $i-1$ to stage i the following recursion is used:

$$\overline{F}_i(q) = \max_{x_i} \{\overline{F}_{i-1}(q - w_i(x_i)) + f_i(x_i) : x_i \text{ is a discretised action of } D_i\} \quad (3.5)$$

Typical for interpolation methods is that when x_i is a discretised action of D_i then $(q - w_i(x_i), i-1)$ is not a discretised state. In order to use the recursion (3.5) an interpolation method uses an interpolation function $\tilde{F}_i(q)$ which interpolates $\overline{F}_i(q)$ at discretised states (q, i) and which is continuous. *i.e.* for discretised states (q, i), $\overline{F}_i(q)$ is calculated by

$$\overline{F}_i(q) = \max_{x_i} \{\tilde{F}_{i-1}(q - w_i(x_i)) + f_i(x_i) : x_i \text{ is a discretised action of } D_i\} \quad (3.6)$$

Having calculated $\overline{F}_i(q)$ for all discretised states of stage i the interpolation function $\tilde{F}_i(q)$ is established, which completes the step of going from stage $i-1$ to stage i. In the sequel it is discussed how FJ action elimination can be applied to interpolation methods.

The maximization in the RHS of (3.6), ignoring for a moment that x_i must be a discretised action, can be reformulated as the following problem:

$$\begin{aligned} \text{maximize} \quad & \tilde{F}_{i-1}(\tilde{q}) + f_i(x_i) && (3.7) \\ \text{subject to} \quad & \tilde{q} + w_i(x_i) = q && (3.8) \end{aligned}$$

In this problem $\tilde{q} \in \mathbb{R}^m$ and $x_i \in \mathbb{R}^{n_i}$ are the free variables. The FJ conditions to this problem are

$$\begin{aligned} \lambda_0 \nabla \tilde{F}_{i-1}(\tilde{q}) &= \lambda, && (3.9) \\ \lambda_0 (\nabla f_i(x_i))^T &= \lambda^T \nabla w_i(x_i), && (3.10) \\ \tilde{q} + w_i(x_i) &= q, && (3.11) \\ (\lambda_0, \lambda) &\neq 0, && (3.12) \\ \lambda_0 &= 0 \quad \text{or} \quad 1 && (3.13) \end{aligned}$$

where $\nabla \tilde{F}_{i-1}(\tilde{q})$ is a m-dimensional gradient column vector, $\nabla f_i(x_i)$ is a n_i-dimensional gradient column vector and $\nabla w_i(x_i)$ is the $m \times n_i$ Jacobian matrix of w_i at x_i. Notice that if $\lambda_0 = 0$ then (3.9) implies $\lambda = 0$ which contradicts (3.12). Hence the conditions (3.9)-(3.13) simplify to

$$\nabla \tilde{F}_{i-1}(\tilde{q}) = \lambda, \tag{3.14}$$

$$(\nabla f_i(x_i))^T = \lambda^T \nabla w_i(x_i), \tag{3.15}$$

$$\tilde{q} + w_i(x_i) = q, \tag{3.16}$$

Next a procedure of an interpolation DP method with FJ action elimination is given for going from stage $i-1$ to stage i for $i \geq 2$. After that this procedure will be explained. For simplicity let's assume that every subproblem corresponding to a state (q,i) is feasible and that for all discretised states $\overline{F}_i(j_1, j_2, ..., j_m) > -\infty$ after the algorithm has completed the step of going from stage $i-1$ to stage i.

for all $(j_1, j_2, ..., j_m) \in \mathbb{Z}^m$ do $\overline{F}_i(j_1, j_2, ..., j_m) := -\infty$

$S_i := \partial D_i \cup \{x : x \in D_i, w_i$ or f_i is not differentiable at $x\}$

if S_i has an infinite number of elements then

let $\overline{S}_i$ be a finite representative set of elements in S_i

else

$\overline{S}_i := S_i$

end if

for all $(j_1, j_2, ..., j_m) \in \mathbb{Z}^m$do

$G := \{q : q \in \mathbb{R}^m, |q_k - j_k \delta_{i-1,k}| \leq \frac{\delta_{i-1,k}}{2} \quad \text{for } 1 \leq k \leq m\}$

define $\underline{\lambda} \in \mathbb{R}^m$ such that for $1 \leq k \leq m : \underline{\lambda}_k \leq \frac{\partial \tilde{F}_{i-1}}{\partial q_k}(q) \quad \forall q \in G$

define $\overline{\lambda} \in \mathbb{R}^m$ such that for $1 \leq k \leq m : \overline{\lambda}_k \geq \frac{\partial \tilde{F}_{i-1}}{\partial q_k}(q) \quad \forall q \in G$

$J := \{x_i : x_i \in D_i \backslash S_i, x_i$ is a discretised action of $D_i, \exists \lambda$ with

$\underline{\lambda} \leq \lambda \leq \overline{\lambda}$ such that λ and x_i satisfy (3.15)$\}$

for all $x \in \overline{S}_i \cup J$ do

$Q := \{q : q \in G, \frac{q_k + w_{i,k}(x)}{\delta_{i,k}} \in \mathbb{Z} \quad \text{for } 1 \leq k \leq m\}$

for all $q \in Q$ do

let $(t_1, t_2, ..., t_m) \in \mathbb{Z}^m$ such that $t_k = \frac{q_k + w_{i,k}(x)}{\delta_{i,k}} \quad \text{for } 1 \leq k \leq m$

if $\overline{F}_i(t_1, t_2, ..., t_m) < \tilde{F}_{i-1}(q) + f_i(x)$ then

$\overline{F}_i(t_1, t_2, ..., t_m) := \tilde{F}_{i-1}(q) + f_i(x)$

$\overline{A}_i(t_1, t_2, ..., t_m) := x$

end if

end do

end do

end do

Discretised states (q, i) are also denoted by $(j_1, j_2, ..., j_m, i)$ where each $j_k \in \mathbb{Z}$ for $1 \leq k \leq m$ and the relationship between (q, i) and $(j_1, j_2, ..., j_m, i)$ is $q_k = j_k \delta_{i,k}$ for $1 \leq k \leq m$. The procedure contains both objects (q, i) and $(j_1, j_2, ..., j_m, i)$. The procedure tries to calculate $\overline{F}_i(j_1, j_2, ..., j_m)$ for every discretised state of stage i. The procedure uses the labelling method (or reaching). It involves not only values of the value function $\overline{F}_{i-1}$ or $\tilde{F}_{i-1}$ at discretised states of stage $i-1$. Therefore, in some sense, the outer loop in the labelling method does not cycle through discretised states of stage $i-1$ but through areas around discretised states of stage $i-1$. These areas around discretised states of stage $i-1$ cover the relevant state space of stage $i-1$. In the procedure this is seen in lines 8 and 9. Line 8 is a loop through discretised states of stage $i-1$ but line 9 defines a box G of states of stage $i-1$ around the discretised states of the embracing loop. Lines 10 and 11 define lower and upper bounds $(\underline{\lambda}, \overline{\lambda})$ for the components of $\nabla \tilde{F}_{i-1}(q)$ on G. Of course, the tighter the bounds are, the greater the potential for action elimination. J is defined as the set of those discretised actions x_i for which (3.15) can hold with a λ satisfying $\underline{\lambda} \leq \lambda \leq \overline{\lambda}$. Discretised actions x_i at which f_i and w_i are differentiable and which are not in J can not be optimal actions to be taken from a state $(q, i-1)$ with $q \in G$ as the FJ conditions can not be fulfilled. Hence only $x_i \in J \cup \overline{S}_i$ need to be considered as actions from states $(q, i-1)$ with $q \in G$. Q (line 15) is the set of those $q \in G$ such that taking the discretised action x from the embracing loop leads to a discretised state of stage i. Usually, interpolation methods have increasing discretisation step lengths for the state space for increasing stages. *i.e.* usually $\delta_{i-1} \leq \delta_i$. Hence typically Q has no element or one element. This means that the loop (for all $q \in Q$ do ...) is a small loop.

When (3.6) is reformulated by (3.7), (3.8) it is ignored that x_i is constrained to be a discretised action of D_i. Therefore the optimality conditions (3.14)-(3.16) are arrived at under the assumption that x_i is a free variable, not constrained to be a discretised action. This means that the algorithm only works well if the density of discretisation points in the action space D_i is large enough, in some sense, in comparison to the density of discretisation points in the state space of stage $i-1$.

3.3 Conclusions

This chapter discussed two effective discretisation schemes for allocation problems of the form (2.1),(2.2) when r is large and showed how to apply FJ action elimination with these schemes.

For large problems with many stages variable grid and interpolation methods are usually the best discretisation schemes. However, interpolation methods have the problem that actions must be interpolated when working backwards the optimal path of states in order to get the solution vector $\widetilde{\boldsymbol{x}}$. This can cause trouble when the problem is highly non-linear or discontinuous at some points. Variable grid methods do not have this problem. Variable grid methods and interpolation methods generally calculate an optimal vector $\widetilde{\boldsymbol{x}}$ which does not satisfy equality constraints exactly since a discretisation error is incurred. There is also this problem with inequality constraints if they should hold as equalities at the solution. To overcome this problem a local optimization procedure can be added at the end of a variable grid or an interpolation method. It is problem dependent whether the variable grid method or the interpolation method is better.

Chapter 4

Allocation problems with multi dimensional state and action space

In this section details of the FJ action elimination will be discussed for problems of the form (2.1) and (2.2) with $m \geq 2$ *i.e.* with more than one constraints. Unlike in sections 2.6 and 3.1, discretised implementations will not be given in full. So far, the objects x_i, w_i were always vectors with dimension n_i, m respectively and it was not necessary to explicitly refer to single components of x_i, w_i. In this section this is necessary and it will be done using a second subscript. For example, the vector x_i has components $x_{i,1}, x_{i,2}..., x_{i,n_i}$. The components of w_i are denoted in similar fashion. Since $m \geq 2$ in this section the vector $\hat{q}$ has components $\hat{q}_1, \hat{q}_2....., \hat{q}_m$. For completeness and easier reference problem (2.1) and (2.2) together with the exact conditions on the variables and functions is stated:

$$\max \sum_{h=1}^{r} f_h(x_h) \tag{4.1}$$

$$\text{subject to} \quad \sum_{h=1}^{r} w_h(x_h) = \hat{q}, \tag{4.2}$$

$$x_h \in D_h \qquad \forall h : 1 \leq h \leq r \tag{4.3}$$

where $D_h \subset I\!R^{n_h}$, $f_h : I\!R^{n_h} \to I\!R$, $w_h : I\!R^{n_h} \to I\!R^m$ and f_h, w_h are piecewise differentiable for $h = 1, ..., r$, and $\hat{q} \in I\!R^m$. The following definitions of objects will make it clear how a practical implementation can be derived from Algorithms A.0.1 or 3.1.2 through some extensions and modifications. These definitions are extensions of Definition 3.1.1. In Definition 3.1.1 (and Algorithm 3.1.2) the state space is one dimensional and hence the discretisation step length of the state space in the $i-$th stage δ_i is a positive real number. Now the state space is $m-$dimensional and therefore δ_i is a $m-$dimensional vector of positive real numbers. As with x_i and w_i, the components of δ_i are indicated using a second

subscript.

Definition 4.0.1

$(\mathbf{j_1}, \mathbf{j_2}, ..., \mathbf{j_m}, \mathbf{i})$ is the state considering $x_1, x_2, ..., x_i$ such that
$(j_k - 1)\delta_{i,k} < \sum_{h=1}^{i} w_{h,k}(x_h) \leq j_k \delta_{i,k}$ for all $k : 1 \leq k \leq m$
and each $x_h \in D_h$ for $1 \leq h \leq i$. For every k, $\quad j_k \in \mathbb{Z}$.

$\mathbf{F_i}(\mathbf{j_1}, \mathbf{j_2}, ..., \mathbf{j_m})$ =maximal value of $\sum_{h=1}^{i} f_h(x_h)$
where $(x_1, x_2, ..., x_i)$ belongs to state $(j_1, j_2, ..., j_m, i)$.

$\overline{\mathbf{F}}_\mathbf{i}(\mathbf{j_1}, \mathbf{j_2}, ..., \mathbf{j_m})$ =computed approximation to $F_i(j_1, j_2, ..., j_m)$.

$\overline{\mathbf{A}}_\mathbf{i}(\mathbf{j_1}, \mathbf{j_2}, ..., \mathbf{j_m})$ =computed value that x_i takes, when $(x_1, x_2, ..., x_i)$
belongs to state $(j_1, j_2, ..., j_m, i)$
and $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j_1, j_2, ..., j_m)$.
i.e. $\overline{A}_i(j_1, j_2, ..., j_m) \in \mathbb{R}^{n_i}$.

$\overline{\mathbf{B}}_\mathbf{i}(\mathbf{j_1}, \mathbf{j_2}, ..., \mathbf{j_m})$ =set of elements (λ_0, λ) such that $\lambda_0 \in \{0, 1\}$,
$\lambda \in \mathbb{R}^m$, $(\lambda_0, \lambda) \neq 0$, if f_h, w_h are differentiable at x_h and $h \leq i$
then $\lambda_0(\nabla f_h(x_h))^T = \lambda^T \nabla w_h(x_h)\}$ where $(x_1, x_2, ..., x_i)$
are the computed vectors which together belong to state
$(j_1, j_2, ..., j_m, i)$ and for which $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j_1, j_2, ..., j_m)$.

$\mathbf{p_i}(\mathbf{j_1}, \mathbf{j_2}, ..., \mathbf{j_m}) = \sum_{h=1}^{i} w_h(x_h)$ where $(x_1, x_2, ..., x_i)$ are the computed
vectors which together belong to state $(j_1, j_2, ..., j_m, i)$
and for which $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j_1, j_2, ..., j_m)$.

Some remarks on these definitions: $(j_1, j_2, ..., j_m, i)$ can be thought of as an infinite set of vectors. $F_i(j_1, j_2, ..., j_m)$ is the solution of a maximization problem with infinite search space. $F_i(j_1, j_2, ..., j_m)$ is approximated by $\overline{F}_i(j_1, j_2, ..., j_m)$. $\overline{A}_i(j_1, j_2, ..., j_m)$ is a real valued action vector. $\overline{B}_i(j_1, j_2, ..., j_m)$ is the set of dual variables (λ_0, λ) which together with the computed optimal solution $x_1, x_2, ..., x_i$ of the subproblem corresponding to state $(j_1, j_2, ..., j_m, i)$ satisfy the FJ conditions. By definition $p_i(j_1, j_2, ..., j_m)$ satisfies:

$$(j_k - 1)\delta_{i,k} < p_{i,k}(j_1, j_2, ..., j_m) \leq j_k \delta_{i,k} \quad \text{for } k = 1, 2, ..., m$$

$p_i(j_1, j_2, ..., j_m)$ is introduced for the correct assignment of $(x_1, x_2, ..., x_i, x_{i+1})$ to states when $(x_1, x_2, ..., x_i)$ are the computed vectors which belong to state $(j_1, j_2, ..., j_m, i)$ and for which $\sum_{h=1}^{i} f_h(x_h) = \overline{F}_i(j_1, j_2, ..., j_m)$.

4.1 Calculating the action set when $n_i = m$

Suppose that for problem (4.1)-(4.3) $n_i = m$ for some $i \leq r$, a FJ-DP algorithm is applied to the problem and the values for $\overline{F}_t, \overline{A}_t, \overline{B}_t, p_t$ have been already calculated for all states $(j_1, j_2, ..., j_m, t)$ with $t \leq i-1$ *i.e.* the FJ-DP algorithm has already advanced to stage $i-1$. Suppose that the discretisation of the state space used is such that in each stage j_k takes z_k different values for $1 \leq k \leq m$ and that $x_i \in D_i$ is discretised such that the component $x_{i,s}$ takes d_s different values for $1 \leq s \leq n_i$, *i.e.* D_i has at most $\prod_{s=1}^{n_i} d_s$ discretisation points. When FJ-DP goes from stage $i-1$ to stage i then it has to calculate the actions which have to be checked from state $(j_1, j_2, ..., j_m, i-1)$. This means for $(\lambda_0, \lambda) \in \overline{B}_{i-1}(j_1, j_2, ..., j_m)$ it has to calculate all $x_i \in D_i$ satisfying

$$\lambda_0 \frac{\partial f_i}{\partial x_{i,s}}(x_i) = \lambda_1 \frac{\partial w_{i,1}}{\partial x_{i,s}}(x_i) + \lambda_2 \frac{\partial w_{i,2}}{\partial x_{i,s}}(x_i) + ... + \lambda_m \frac{\partial w_{i,m}}{\partial x_{i,s}}(x_i) \qquad (4.4)$$
$$\text{for all } s : 1 \leq s \leq n_i$$

The quick solution of the above n_i simultaneous equations in x_i depends a lot on problem specific information available. If no additional information is available and the solution has to be found numerically the question is whether FJ-DP is any faster than ordinary DP.

Suppose the solutions to the above equations have to be found numerically. Let's discuss how this can be done such that FJ-DP is faster than ordinary DP. For this it is assumed that the functions f_i, w_i are piecewise continuously differentiable, not just piecewise differentiable. One good method would be the following: Before doing the DP step of going from stage $i-1$ to stage i run once through the discretised actions x_i to get additional information. The two cases $\lambda_0 = 0$ and $\lambda_0 = 1$ have to be looked at separately. Let's first do the case $\lambda_0 = 0$. Look at

$$0 = \hat{\lambda}_1 \frac{\partial w_{i,1}}{\partial x_{i,s}}(x_i) + \hat{\lambda}_2 \frac{\partial w_{i,2}}{\partial x_{i,s}}(x_i) + ... + \hat{\lambda}_m \frac{\partial w_{i,m}}{\partial x_{i,s}}(x_i) \quad \text{for } s = 1, 2, ..., n_i \qquad (4.5)$$

for each discretisation point $x_i \in D_i$ and decide whether these n_i $(= m)$ linear equations in $\hat{\lambda}_1, \hat{\lambda}_2, ..., \hat{\lambda}_m$ allow a solution $(\hat{\lambda}_1, \hat{\lambda}_2, ..., \hat{\lambda}_m) \neq 0$ or almost allow such a solution or do not allow such a solution. If they allow such a solution include this point x_i in a set P_0 which initially is empty. If they almost allow such a solution then use a local search method like Newton's Method to find a close point which satisfies (4.5) for some $(\hat{\lambda}_1, \hat{\lambda}_2, ..., \hat{\lambda}_m) \neq 0$ and include this point in P_0. It is expected that the number of elements of P_0 is at least one order of magnitude less than the number of discretisation points $x_i \in D_i$.

For the case $\lambda_0 = 1$ solve the following n_i $(= m)$ simultaneous linear equations in $\tilde{\lambda}_1, \tilde{\lambda}_2, ..., \tilde{\lambda}_m$ for every discretised action x_i if possible and if there is a unique solution.

$$\frac{\partial f_i}{\partial x_{i,s}}(x_i) = \tilde{\lambda}_1 \frac{\partial w_{i,1}}{\partial x_{i,s}}(x_i) + \tilde{\lambda}_2 \frac{\partial w_{i,2}}{\partial x_{i,s}}(x_i) + ... + \tilde{\lambda}_m \frac{\partial w_{i,m}}{\partial x_{i,s}}(x_i) \qquad (4.6)$$
$$\text{for all } s : 1 \leq s \leq n_i$$

In this way a finite number of one-dimensional search intervals of maximal length of the form

$$\tilde{I}_t := \{(a + \theta(b-a), c_2, c_3, ..., c_{n_i}) : 0 \leq \theta \leq 1\} \qquad (4.7)$$

can be established such that $a, b \in I\!R$, c_l is a discretised value of the component $x_{i,l}$ for $2 \leq l \leq n_i$, the functions f_i, w_i are differentiable on $\tilde{I}_t$ and $\tilde{\lambda}_1$ is increasing or decreasing on $\tilde{I}_t$.

Ordinary DP simply checks every discretised action $x_i \in D_i$ as a possibly optimal action to be taken from state $(j_1, j_2, ..., j_m, i-1)$. This requires $O(\prod_{k=1}^{n_i} d_k)$ operations. Assuming that $B_{i-1}(j_1, j_2, ..., j_m)$ has only one element (λ_0, λ) FJ-DP only checks those values of $x_i \in D_i$ satisfying (4.4) as possibly optimal actions to be taken from state $(j_1, j_2, ..., j_m, i-1)$. Assuming that the space of points $x_i \in D_i$ where f_i or w_i is not differentiable has at least one dimension less than the set D_i itself, if $\lambda_0 = 1$ then FJ-DP can find the points $x_i \in D_i$ approximately satisfying (4.4) and having a discretised value for the component $x_{i,l}$ where $2 \leq l \leq n_i$ with expectedly $O(\log d_1 \prod_{k=2}^{n_i} d_k)$ operations since on the intervals $\tilde{I}_t$ binary bisection can be used to find a point x_i with $\tilde{\lambda}_1$ being equal to λ_1. The establishing of the search intervals $\tilde{I}_t$ causes $O(\prod_{k=1}^{n_i} d_k)$ amount of work. If $\lambda_0 = 0$ then FJ-DP only has to check the actions $x_i \in P_0$ as possibly optimal actions to be taken from state $(j_1, j_2, ..., j_m, i-1)$. Since P_0 is most likely to have much fewer than $\prod_{k=1}^{n_i} d_k$ points, FJ-DP does less work than ordinary DP when $\lambda_0 = 0$. However, the case $\lambda_0 = 0$ is usually very rare. Hence FJ-DP is expected to take $O(\prod_{k=1}^{n_i} d_k)$ $+ O(\prod_{k=1}^{m} z_k \log d_1 \prod_{k=2}^{n_i} d_k)$ operations to go from stage $i-1$ to stage i, Ordinary DP takes $O(\prod_{k=1}^{m} z_k \prod_{k=1}^{n_i} d_k)$ operations for this.

If among $\{d_1, d_2, ..., d_{n_i}\}$ d_l is the largest it is a good idea to establish the search intervals $\tilde{I}_t$ along the axis of $x_{i,l}$ instead of $x_{i,1}$. In this way the complexity is reduced by making better use of the binary bisection search.

When giving the complexity of FJ-DP the 'expected complexity' was given. This is because two assumptions were necessary. Firstly that the number of states $(j_1, j_2, ..., j_m, i-1)$ of stage $i-1$ for which $B_{i-1}(j_1, j_2, ..., j_m)$ has more than one element is of magnitude less than the total number of states of stage $i-1$ (which is at most $\prod_{k=1}^{m} z_k$), secondly that the space of points $x_i \in D_i$ where f_i or w_i is not differentiable has at least one dimension less than the set D_i itself.

4.2 Finding the action set when $n_i > m$

Suppose that for problem (4.1)-(4.3) $n_i > m$ for some $i \leq r$, a FJ-DP algorithm is applied to the problem and the values for $\overline{F}_t, \overline{A}_t, \overline{B}_t, p_t$ have been already calculated for all states $(j_1, j_2, ..., j_m, t)$ with $t \leq i-1$. Suppose that the discretisation of the state space and action space is the same as in subsection 4.1. When FJ-DP goes from stage $i-1$ to stage i then it has to find the actions which have to be checked from state $(j_1, j_2, ..., j_m, i-1)$. This means for $(\lambda_0, \lambda) \in \overline{B}_{i-1}(j_1, j_2, ..., j_m)$ it has to find all $x_i \in D_i$ satisfying (4.4). If this can be done analytically, good. In the sequel it is shown how this can be done numerically such that FJ-DP is faster than ordinary DP.

Before doing the DP step of going from stage $i-1$ to stage i run once through the discretised actions x_i to get additional information. The two cases $\lambda_0 = 0$ and $\lambda_0 = 1$ have to be looked at separately. For the case $\lambda_0 = 0$ look at (4.5) for each discretisation point $x_i \in D_i$ and decide whether these n_i $(> m)$ linear equations in $\hat{\lambda}_1, \hat{\lambda}_2, ..., \hat{\lambda}_m$ allow a solution $(\hat{\lambda}_1, \hat{\lambda}_2, ..., \hat{\lambda}_m) \neq 0$ or almost allow such a solution or do not allow such a solution. If they allow such a solution include this point x_i in a set P_0 which initially is empty. If they almost allow such a solution then use a local search method like Newton's Method to try to find a close point which satisfies (4.5) for some $(\hat{\lambda}_1, \hat{\lambda}_2, ..., \hat{\lambda}_m) \neq 0$ and include this point in P_0.

For the case $\lambda_0 = 1$ look at (4.6) for each discretisation point $x_i \in D_i$ and, as before, decide whether these n_i $(> m)$ linear equations in $\tilde{\lambda}_1, \tilde{\lambda}_2, ..., \tilde{\lambda}_m$ allow a solution or almost allow a solution or do not allow a solution. If they allow a solution include this point x_i in a set P_1 which initially is empty. If they almost allow a solution then use a local search method like Newton's Method to try to find a close point x_i which satisfies (4.6) for some $\tilde{\lambda}_1, \tilde{\lambda}_2, ..., \tilde{\lambda}_m$ and include this point in P_1.

It is expected that the number of elements of P_0 and of P_1 is at least one order of magnitude less than the number of discretisation points $x_i \in D_i$. Instead of checking every discretisation point $x_i \in D_i$ as a possibly optimal action from state $(j_1, j_2, ..., j_m, i-1)$ only the points in P_0 are checked if $\lambda_0 = 0$ and only the points in P_1 are checked if $\lambda_0 = 1$ when (λ_0, λ) is the only element of $B_{i-1}(j_1, j_2, ..., j_m)$. Hence when the action set is found numerically FJ-DP is expected to need $O(\prod_{k=1}^{n_i} d_k)$ operations to establish the sets P_0, P_1 and $O(\prod_{k=1}^{m} z_k \max(|P_0|, |P_1|))$ operations to calculate $\overline{F}_i, \overline{A}_i, \overline{B}_i, p_i$ for all states of stage i.

4.3 Calculating the action set when $n_i < m$

Suppose that for problem (4.1)-(4.3) $n_i < m$ for some $i \leq r$, a FJ-DP algorithm is applied to the problem and the values for $\overline{F}_t, \overline{A}_t, \overline{B}_t, p_t$ have been already calculated for all states $(j_1, j_2, ..., j_m, t)$ with $t \leq i-1$. When FJ-DP goes from stage $i-1$ to stage i then it has to calculate the actions which have to be checked from state $(j_1, j_2, ..., j_m, i-1)$. This means for $(\lambda_0, \lambda) \in \overline{B}_{i-1}(j_1, j_2, ..., j_m)$ it has to calculate all $x_i \in D_i$ satisfying (4.4). A good numerical procedure to do this has not been found for the case that no special information about the functions f_i and w_i can be exploited. One problem is that when $\lambda_0 = 1$ then (4.6) is a non-homogeneous set of n_i $(< m)$ linear equation in $\lambda_1, ..., \lambda_m$ which has more variables than equations *i.e.* usually (4.6) allows infinitely many solutions $(\tilde{\lambda}_1, \tilde{\lambda}_2, ..., \tilde{\lambda}_m)$ for every discretisation point x_i. This makes it hard to exploit information collected by running initially through the discretised actions $x_i \in D_i$ in order to then use this information to speed up the calculations of actions which have to be checked from states $(j_1, j_2, ..., j_m, i-1)$.

4.4 A special case for $n_i < m$

In the last subsection the difficulty was outlined of finding a good numerical procedure for calculating the actions which have to be checked from state $(j_1, j_2, ..., j_m, i-1)$ when FJ-DP goes from stage $i-1$ to stage i, $n_i < m$ and no special structure of the functions f_i and w_i can be exploited. One kind of structure of the functions f_i, w_i will be discussed in this section. This structure allows a good numerical procedure for calculating the actions which have to be checked from state $(j_1, j_2, ..., j_m, i-1)$ when FJ-DP goes from stage $i-1$ to stage i.

Suppose that f_i and w_i linearly depend on k functions $g_{i,1}(x_i)$, $g_{i,2}(x_i)$,..., $g_{i,k}(x_i)$ and $g_{i,j}(x_i) \in \mathbb{R}$ for $1 \leq j \leq k$. Let $g_i(x_i) \in \mathbb{R}^k$ with components $g_{i,1}(x_i)$, $g_{i,2}(x_i)$,..., $g_{i,k}(x_i)$ and

$$f_i(x_i) = c_i^T g_i(x_i), \tag{4.8}$$

$$w_i(x_i) = A_i g_i(x_i) \tag{4.9}$$

$$k \leq n_i + 1 \tag{4.10}$$

where $c_i \in \mathbb{R}^k$ and A_i is an $m \times k$ matrix. Inserting these relationships into (4.4) yields equations of the form

$$\mu_1 \frac{\partial g_{i,1}}{\partial x_{i,s}}(x_i) = \mu_2 \frac{\partial g_{i,2}}{\partial x_{i,s}}(x_i) + \mu_3 \frac{\partial g_{i,3}}{\partial x_{i,s}}(x_i) + ... + \mu_k \frac{\partial g_{i,k}}{\partial x_{i,s}}(x_i) \tag{4.11}$$

for all $s : 1 \leq s \leq n_i$

where μ_1, μ_2,...,μ_k are all depending on c_i, A_i and (λ_0, λ). When FJ-DP goes from stage $i-1$ to stage i then it has to calculate the actions which have to be checked from state $(j_1, j_2, ..., j_m, i-1)$. This means for $(\lambda_0, \lambda) \in \overline{B}_{i-1}(j_1, j_2, ..., j_m)$ it has to calculate all $x_i \in D_i$ satisfying (4.4). In our case these actions can be found by calculating all $x_i \in D_i$ satisfying (4.11) after having computed μ_1, μ_2,..., μ_k from c_i, A_i and (λ_0, λ).

If $k = n_i + 1$ the actions $x_i \in D_i$ satisfying (4.11) for a given $(\lambda_0, \lambda) \in \overline{B}_{i-1}(j_1, j_2, ..., j_m)$ can be found quickly when the functions $g_{i,1}$, $g_{i,2}$,..., $g_{i,k}$ are investigated before doing the DP recursion of going from stage $i-1$ to stage i. The idea is to use the method described in section 4.1 for the finding of all $x_i \in D_i$ satisfying (4.11). The method described in section 4.1 finds all $x_i \in D_i$ satisfying (4.4) for a given (λ_0, λ). In section 4.1 $m = n_i$ hence (4.4) consists of n_i equations, each containing $n_i + 1$ dual variables (namely λ_0, λ_1,..., λ_m) and $n_i + 1$ functions (namely f_i, $w_{i,1}$, $w_{i,2}$,..., $w_{i,m}$). Compare this to (4.11) which also consists of n_i equations, each containing $n_i + 1$ dual variables (namely μ_1, μ_2,..., μ_k) and $n_i + 1$ functions (namely $g_{i,1}$, $g_{i,2}$,..., $g_{i,k}$). In section 4.1 the method described there deals separately with the cases $\lambda_0 = 0$ and $\lambda_0 = 1$. The method for finding all $x_i \in D_i$ satisfying (4.4) when $\lambda_0 = 0$ in section 4.1 can be applied for the finding of all $x_i \in D_i$ satisfying (4.11) when $\mu_1 = 0$. The method for finding all $x_i \in D_i$ satisfying (4.4) when $\lambda_0 = 1$ in section 4.1 can be applied for the finding of all $x_i \in D_i$ satisfying

$$\frac{\partial g_{i,1}}{\partial x_{i,s}}(x_i) = \frac{\mu_2}{\mu_1}\frac{\partial g_{i,2}}{\partial x_{i,s}}(x_i) + \frac{\mu_3}{\mu_1}\frac{\partial g_{i,3}}{\partial x_{i,s}}(x_i) + ... + \frac{\mu_k}{\mu_1}\frac{\partial g_{i,k}}{\partial x_{i,s}}(x_i) \qquad (4.12)$$
$$\text{for all } s : 1 \leq s \leq n_i$$

when $\mu_1 \neq 0$. But when $\mu_1 \neq 0$ then x_i satisfies (4.12) if and only if it satisfies (4.11). Hence, when $k = n_i + 1$ then the method of section 4.1, described there for the finding of all $x_i \in D_i$ satisfying (4.4) given (λ_0, λ), can also be used for the finding of all $x_i \in D_i$ satisfying (4.11) given (λ_0, λ).

If $k < n_i + 1$ then the actions $x_i \in D_i$ satisfying (4.11) for a given (λ_0, λ) can be obtained using the method of section 4.2, which finds the actions $x_i \in D_i$ satisfying (4.4) for given (λ_0, λ) when $n_i > m$. The argument for the suitability of the method described in section 4.2 is similar to the argument of the previous paragraph.

The example of section 2.7 has the structure described by (4.8),(4.9) which will be explained in the sequel. In problem (2.41) f_h is not used in the same way as it is used in (4.1)-(4.3). In order to avoid confusion let's restate the problem (2.41) renaming f_h there with s_h. Let's also replace the dummy variable h by i.

So problem (2.41) has the form

$$\begin{aligned} \text{maximize} & \sum_{i=1}^{r} \frac{i}{2i+1} s_i(x_i) \\ \text{subject to} & \sum_{i=1}^{r} \frac{i+1}{2i+1} s_i(x_i) = \hat{q}_1, \\ & \sum_{i=1}^{r} x_i = \hat{q}_2, \\ & x_i \in I\!R, \quad x_i \in [0, d_i] \qquad 1 \le i \le r, \end{aligned} \tag{4.13}$$

with s_i being functions $I\!R \to I\!R$, $\hat{q}_1 \in I\!R$, $\hat{q}_2 \in I\!R$. Now, problem (4.13) corresponds to the general problem form (4.1)-(4.3) by setting

$$f_i(x_i) = \frac{i}{2i+1} s_i(x_i), \tag{4.14}$$

$$w_i(x_i) = \begin{pmatrix} \frac{i+1}{2i+1} s_i(x_i) \\ x_i \end{pmatrix}. \tag{4.15}$$

(4.14),(4.15) correspond to the form (4.8),(4.9) by setting

$$\begin{aligned} c_i &= \begin{pmatrix} \frac{i}{2i+1} \\ 0 \end{pmatrix}, \\ g_i(x_i) &= \begin{pmatrix} s_i(x_i) \\ x_i \end{pmatrix}, \\ A_i &= \begin{pmatrix} \frac{i+1}{2i+1} & 0 \\ 0 & 1 \end{pmatrix}. \end{aligned}$$

Since here $k = 2$, $n_i = 1$ and $m = 2$ we have $k = n_i + 1$. Hence the problem of section 2.7 possesses the structure (4.8)-(4.10).

4.5 Conclusions

In FJ-DP it is very important to have a quick method to find all actions $x_i \in D_i$ that satisfy (2.10) or equivalently (4.4) for any given (λ_0, λ). The speed-up achieved by the action elimination essentially depends on this. In this chapter it was discussed how to numerically find the actions satisfying (4.4) such that the action elimination results in a speed-up of DP.

For the cases $n_i = m$ and $n_i > m$ methods have been shown which solve (4.4) fast enough to make FJ-DP faster than ordinary DP. For the case $n_i < m$ a good numerical way for solving (4.4) has only been found for the special situation when the functions f_i and w_i linearly depend on at most $n_i + 1$ linearly independent functions $I\!R^{n_i} \to I\!R$.

Chapter 5

Discrete problems

In this section discrete allocation problems will be discussed and it will be shown how the FJ action elimination can be applied. The class of problems considered is:

$$\max \sum_{h=1}^{r} f_h(x_h) \tag{5.1}$$

$$\text{subject to} \quad \sum_{h=1}^{r} w_h(x_h) = \hat{q}, \tag{5.2}$$

$$x_h \in D_h \qquad \text{for } h = 1, 2, ..., r \tag{5.3}$$

where $D_h \subset \mathbb{Z}^{n_h}$, $f_h : \mathbb{Z}^{n_h} \to \mathbb{R}$, $w_h : \mathbb{Z}^{n_h} \to \mathbb{R}^m$ for $h = 1, ..., r$, and $\hat{q} \in \mathbb{R}^m$.

The FJ conditions are first order optimality conditions for differentiable functions but the class of problems above involves discrete and therefore not differentiable functions. The FJ action elimination can be applied by using interval arithmetic for derivative approximations. This point needs further explanation.

The FJ conditions employ partial derivatives. Since f_i, w_i are only defined for discrete points $x_i \in \mathbb{Z}^{n_i}$ there are only forward and backward differences. However, the view is taken that there always exist continuous and differentiable interpolating functions $\tilde{f}_i : \mathbb{R}^{n_i} \to \mathbb{R}$ and $\tilde{w}_i : \mathbb{R}^{n_i} \to \mathbb{R}^m$ such that

a) the value of f_i coincides with that of $\tilde{f}_i$ and the value of w_i coincides with that of $\tilde{w}_i$ for all arguments $x_i \in \mathbb{Z}^{n_i}$,

b) the partial derivatives of $\tilde{f}_i$ and $\tilde{w}_i$ at $x_i \in \mathbb{Z}^{n_i}$ are bounded by the forward and backward difference approximations with step length 1.

The requirement b) will be explained. For $x_i \in \mathbb{Z}^{n_i}$ $\frac{\partial \tilde{f}_i}{\partial x_{i,k}}(x_i)$ (where $x_{i,k}$ is again the k-th component of vector x_i) has the forward difference approximation of step length 1 $f_i(x_i + e_k) - f_i(x_i)$ where e_k is the k-th unit vector *i.e.* the

vector with the k-th component equal to 1 and the other components equal to 0. Similarly the backward difference approximation is $f_i(x_i) - f_i(x_i - e_k)$. For $x_i \in D_i$, $(x_i + e_k) \in D_i$, $(x_i - e_k) \in D_i$ let

$$\begin{aligned}\overline{a} &:= \max\{f_i(x_i + e_k) - f_i(x_i), \quad f_i(x_i) - f_i(x_i - e_k)\},\\ \underline{a} &:= \min\{f_i(x_i + e_k) - f_i(x_i), \quad f_i(x_i) - f_i(x_i - e_k)\},\end{aligned}$$

if $x_i \in D_i$, $(x_i + e_k) \in D_i$, $(x_i - e_k) \notin D_i$ let

$$\begin{aligned}\overline{a} &:= \infty,\\ \underline{a} &:= f_i(x_i + e_k) - f_i(x_i),\end{aligned}$$

if $x_i \in D_i$, $(x_i + e_k) \notin D_i$, $(x_i - e_k) \in D_i$ let

$$\begin{aligned}\overline{a} &:= f_i(x_i) - f_i(x_i - e_k),\\ \underline{a} &:= -\infty,\end{aligned}$$

and if $x_i \in D_i$, $(x_i + e_k) \notin D_i$, $(x_i - e_k) \notin D_i$ let

$$\begin{aligned}\overline{a} &:= \infty,\\ \underline{a} &:= -\infty,\end{aligned}$$

(The definitions of $\overline{a}$ and $\underline{a}$ are achieved in the cases $(x_i + e_k) \notin D_i$ and $(x_i - e_k) \notin D_i$ by thinking of $f_i(x_i + e_k)$ and $f_i(x_i - e_k)$ respectively as being equal to $-\infty$.) $\frac{\partial f_i}{\partial x_{i,k}}$ does not exist in the classical meaning of partial derivatives since f_i is a discrete and hence not a differentiable function. Define

$$\frac{\partial f_i}{\partial x_{i,k}}(x_i) := [\underline{a}, \overline{a}] \tag{5.4}$$

i.e. $\frac{\partial f_i}{\partial x_{i,k}}(x_i)$ is an interval with the boundaries being determined by the forward and backward difference approximations to the classical partial derivative. The requirement b) above for the partial derivative $\frac{\partial \tilde{f}_i}{\partial x_{i,k}}(x_i)$ is simply:

$$\frac{\partial \tilde{f}_i}{\partial x_{i,k}}(x_i) \in \frac{\partial f_i}{\partial x_{i,k}}(x_i)$$

The intervals bounding the partial derivatives $\frac{\partial \bar{w}_{i,j}}{\partial x_{i,k}}(x_i)$ can be found in similar but not identical way. Let

$$\begin{aligned}\overline{a} &:= \max\{w_{i,j}(x_i + e_k) - w_{i,j}(x_i), w_{i,j}(x_i) - w_{i,j}(x_i - e_k)\},\\ \underline{a} &:= \min\{w_{i,j}(x_i + e_k) - w_{i,j}(x_i), w_{i,j}(x_i) - w_{i,j}(x_i - e_k)\},\end{aligned}$$

(Note that this definition for $\overline{a}$ and $\underline{a}$ is irrespective of $x_i \in D_i$ or $x_i \notin D_i$, unlike in the previous definition leading to $\frac{\partial f_i}{\partial x_{i,k}}(x_i)$.) Define

$$\frac{\partial w_{i,j}}{\partial x_{i,k}}(x_i) := [\underline{a}, \overline{a}].$$

$\frac{\partial w_{i,j}}{\partial x_{i,k}}(x_i)$ for $x_i \in \mathbb{Z}^{n_i}$ be the bounding interval for $\frac{\partial \tilde{w}_{i,j}}{\partial x_{i,k}}(x_i)$ where $\tilde{w}_{i,j}$ is, again, the j-th component of vector $\tilde{w}_i$. In the sequel it is described how a discrete version of FJ action elimination is applied to problem (5.1)-(5.3). The description will be theoretical (like in sections 2.1-2.5) in the sense that the state space will be continuous. (In sections 2.1-2.5 the state space and action space were continuous, but problem (5.1)-(5.3) imposes a discrete action set.)

For DP applied to problem (5.1)-(5.3) the subproblem corresponding to state (q, i) is

$$F_i(q) \equiv \max_{x_1, x_2, \ldots, x_i} \sum_{h=1}^{i} f_h(x_h) \tag{5.5}$$

$$\text{subject to } \sum_{h=1}^{i} w_h(x_h) = q, \tag{5.6}$$

$$x_h \in D_h \text{ for } h = 1, 2, \ldots, i \tag{5.7}$$

where $q \in \mathbb{R}^m$ and all other quantities are as in (5.1)-(5.3). If this subproblem is infeasible then define $F_i(q) := -\infty$. The DP recursion is

$$F_i(q) = \max_{x_i \in D_i} \{f_i(x_i) + F_{i-1}(q - w_i(x_i))\}. \tag{5.8}$$

For the discrete problem the meaning of $B_i(q, \boldsymbol{x}_i)$, where $\boldsymbol{x}_i^T = (x_1^T, x_2^T, \ldots, x_i^T))$, is different from that for continuous problems which was described in section 2.2. Let $\nabla f_h(x_h)$ be the n_h-dimensional column vector of intervals where the k-th component is $\frac{\partial f_h}{\partial x_{h,k}}(x_h)$. Let $\nabla w_h(x_h)$ be the $m \times n_h$ matrix of intervals where the (j,k)-th entry is $\frac{\partial w_{h,j}}{\partial x_{h,k}}(x_h)$. The set $B_i(q, \boldsymbol{x}_i)$ is the set of (λ_0, λ) where $\lambda_0 = 0$ or 1 and $\lambda \in \mathbb{R}^m$ which satisfy

$$\lambda_0(\nabla f_h(x_h))^T \cap \lambda^T \nabla w_h(x_h) \neq \emptyset, \text{ for } h = 1, \ldots, i, \tag{5.9}$$

$$\sum_{h=1}^{i} w_h(x_h) = q, \tag{5.10}$$

$$\lambda_0 = 0 \text{ or } \lambda_0 = 1, \tag{5.11}$$

$$(\lambda_0, \lambda) \neq 0 \tag{5.12}$$

In (5.9) the intersection sign applies to the k-th component of the vector left of it and to the k-th component of the vector right of it for every $k : 1 \leq k \leq n_h$. The

not-equality sign in (5.9) applies to every component of the LHS n_h-dimensional vector of intervals. More explicitly, (5.9) says that

$$\lambda_0 \frac{\partial f_h}{\partial x_{h,k}}(x_h) \cap \left(\sum_{j=1}^{m} \lambda_j \frac{\partial w_{h,j}}{\partial x_{h,k}}(x_h) \right) \neq \emptyset \qquad (5.13)$$
$$\text{for every } 1 \leq h \leq i, \quad 1 \leq k \leq n_h$$

where the expression in big brackets and the expression $\lambda_0 \frac{\partial f_h}{\partial x_{h,k}}(x_h)$ are evaluated according to the rules of interval arithmetic. Let $\boldsymbol{x}_{i-1}$ be like $\boldsymbol{x}_i$ except that the last component is missing, *i.e.* $\boldsymbol{x}_{i-1}^T = (x_1^T, x_2^T \ldots x_{i-1}^T)$. For $i \geq 2$ the set of properties in (5.9)-(5.12) defining the set $B_{i-1}(q - w_i(x_i), \boldsymbol{x}_{i-1})$ is fully contained in the set of properties defining $B_i(q, \boldsymbol{x}_i)$. Furthermore the properties which are amongst those defining $B_i(q, \boldsymbol{x}_i)$ but not amongst those defining $B_{i-1}(q - w_i(x_i), \boldsymbol{x}_{i-1})$ are

$$\lambda_0(\nabla f_i(x_i))^T \cap \lambda^T \nabla w_i(x_i) \neq \emptyset \qquad (5.14)$$

Let $N_i(x_i)$ be the set of all (λ_0, λ) which satisfy (5.14). It follows that for $i \geq 2$

$$B_i(q, \boldsymbol{x}_i) = B_{i-1}(q - w_i(x_i), \boldsymbol{x}_{i-1}) \cap N_i(x_i).$$

The DP action elimination algorithm employs a function of states $B_i(q)$ defined by

$$B_i(q) = B_i(q, \tilde{\boldsymbol{x}}_i),$$

where $\tilde{\boldsymbol{x}}_i$ is the optimal vector for subproblem (q, i) that the algorithm has computed at the time of defining $B_i(q)$. The essence of the action elimination is to replace (5.8) with the recursion below.

$$F_i(q) = \max_{x_i \in D_i} \{ f_i(x_i) + F_{i-1}(q - w_i(x_i)) : \exists (\lambda_0, \lambda) \in B_{i-1}(q - w_i(x_i))$$
$$\text{such that } (\lambda_0, \lambda), x_i \text{ satisfy (5.14)} \} \qquad (5.15)$$

The argument for this action elimination for discrete problems of the form (5.1)-(5.3) is the following: suppose that $\tilde{f}_h$ and $\tilde{w}_h$ are defined for $x_h \in I\!R^{n_h}$ and are interpolating functions for f_h and w_h respectively. Further suppose that the problem

$$\max \sum_{h=1}^{r} \tilde{f}_h(x_h)$$
$$\text{subject to} \quad \sum_{h=1}^{r} \tilde{w}_h(x_h) = \hat{q},$$

were solved by using DP with (continuous) FJ action elimination (as described in sections 2.1-2.5) but the action space has been discretised and the set of discretised

actions x_h corresponds to D_h. The solution of this discretised DP problem is the same as the solution of (5.1)-(5.3) since $\tilde{f}_h$, $\tilde{w}_h$ are the interpolating functions of f_h and w_h respectively. Applying the FJ conditions (2.6)- (2.9) in state (q,i) for the functions $\tilde{f}_h$, $\tilde{w}_h$ together with the uncertainty about the partial derivatives of $\tilde{f}_h$, $\tilde{w}_h$ expressed by

$$\frac{\partial \tilde{f}_h}{\partial x_{h,k}}(x_h) \in \frac{\partial f_h}{\partial x_{h,k}}(x_h) \text{ for all } 1 \leq k \leq n_h$$

$$\text{and} \qquad \frac{\partial \tilde{w}_{h,j}}{\partial x_{h,k}}(x_h) \in \frac{\partial w_{h,j}}{\partial x_{h,k}}(x_h) \text{ for all } 1 \leq k \leq n_h, 1 \leq j \leq m$$

yields the conditions (5.9)- (5.12). These conditions in turn suggest the replacement of (5.8) with (5.15). In order to implement the recursion (5.15) for going from stage $i-1$ to stage i (for $i \geq 2$) Procedure 2.3.1 of section 2.3 is used with the third command (lines 3 and 4) being replaced with

for all $x_i \in D_i$ for which $\exists(\lambda_0, \lambda) \in B_{i-1}(q)$
such that $(\lambda_0, \lambda), x_i$ satisfy (5.14) do

Conditions (5.9)-(5.12) define the set $B_i(q, \boldsymbol{x}_i)$. When implementing this action elimination in practice the question is how to obtain a more useful description or classification for the set $B_i(q, \boldsymbol{x}_i)$ than (5.9)-(5.12). Even to work with a set $\tilde{B}_i(q, \boldsymbol{x}_i)$ which is a superset of $B_i(q, \boldsymbol{x}_i)$ might be preferable to working with $B_i(q, \boldsymbol{x}_i)$ if the description of $\tilde{B}_i(q, \boldsymbol{x}_i)$ is simple.

If $\sum_{h=1}^{i} n_h \geq m$ then the number of interval relations in (5.9), or equivalently in (5.13), is at least as big as the number of components of vector λ. Consider the following set of linear interval-arithmetic equations in (λ_0, λ):

$$\lambda_0 \frac{\partial f_h}{\partial x_{h,k}}(x_h) = \sum_{j=1}^{m} \lambda_j \frac{\partial w_{h,j}}{\partial x_{h,k}}(x_h) \qquad (5.16)$$
$$\text{for every } 1 \leq k \leq n_h, \quad 1 \leq h \leq i$$

It is possible to deduce from this set of equations confidence intervals for λ for the two cases $\lambda_0 = 0$ and $\lambda_0 = 1$ using Gaussian elimination in interval arithmetic. In this way it is possible to calculate confidence regions for every component of λ for the case $\lambda_0 = 0$ and for the case $\lambda_0 = 1$ which define a superset $\tilde{B}_i(q, \boldsymbol{x}_i)$ for $B_i(q, \boldsymbol{x}_i)$.

The following procedure is a practical implementation for going from stage $i-1$ to stage i. Let

$$\tilde{B}_i(q) = \tilde{B}_i(q, \tilde{\boldsymbol{x}}_i)$$

where $\widetilde{\boldsymbol{x}}_i$ is the vector which defines $B_i(q)$ *i.e.* for which

$$B_i(q) = B_i(q, \widetilde{\boldsymbol{x}}_i).$$

$\widetilde{\boldsymbol{x}}_i$ is the optimal vector for subproblem (q, i) that the algorithm has computed at the time of defining $B_i(q)$. The idea behind the following procedure is to not use action elimination if $\sum_{h=1}^{i-1} n_h < m$, to compute $\tilde{B}_i(q)$ for the first time if $\sum_{h=1}^{i-1} n_h \geq m$ and $\sum_{h=1}^{i-2} n_h < m$, and thirdly, to use action elimination when $\sum_{h=1}^{i-1} n_h \geq m$.

for all $q \in I\!R^m$ set $F_i(q) := -\infty$
if $\sum_{h=1}^{i-1} n_h < m$ then
  for all $q \in I\!R^m$ for which $F_{i-1}(q) > -\infty$ do
    for all $x_i \in D_i$ do
      if $F_i(q + w_i(x_i)) < F_{i-1}(q) + f_i(x_i)$ then
        $F_i(q + w_i(x_i)) := F_{i-1}(q) + f_i(x_i)$
        $A_i(q + w_i(x_i)) := x_i$
      end if
    end do x_i
  end do q
else (if $\sum_{h=1}^{i-1} n_h \geq m$)
  for all $q \in I\!R^m$ for which $F_{i-1}(q) > -\infty$ do
    if $\sum_{h=1}^{i-2} n_h < m$ then calculate $\tilde{B}_{i-1}(q)$
    for all $x_i \in D_i$ for which $\exists(\lambda_0, \lambda) \in \tilde{B}_{i-1}(q)$
        such that $(\lambda_0, \lambda), x_i$ satisfy (5.14) do
      if $F_i(q + w_i(x_i)) < F_{i-1}(q) + f_i(x_i)$ then
        $F_i(q + w_i(x_i)) := F_{i-1}(q) + f_i(x_i)$
        $A_i(q + w_i(x_i)) := x_i$
        $\tilde{B}_i(q + w_i(x_i)) := \tilde{B}_{i-1}(q) \cap N_i(x_i)$
      end if
    end do x_i
  end do q
end if

5.1 Finding the action set

As for the continuous case FJ action elimination, for the discrete case action elimination DP algorithm it is important to find the action set corresponding to a state $(q, i-1)$ and set $\tilde{B}_{i-1}(q)$ quickly. The action set now are those $x_i \in D_i$ for which (5.14) is fulfilled for some $(\lambda_0, \lambda) \in \tilde{B}_{i-1}(q)$. The calculation of the action set has to be done numerically. Methods analogous to those in subsections 4.1, 4.2 and 4.3 can be used. The main adaptation necessary is to use interval arithmetic instead of real arithmetic when solving systems of linear equations (4.5) or (4.6). In the case $n_i = m$ the search intervals $\tilde{I}_t$ in (4.7) have to change their properties. In the continuous case $\tilde{I}_t$ are such that $\tilde{\lambda}_1$ is increasing or decreasing. Such a meaning of $\tilde{I}_t$ depends on $\tilde{\lambda}_1 \in I\!R$. In the discrete case, after having solved (4.6) for $\tilde{\lambda}$ using interval arithmetic, there are only confidence intervals for $\tilde{\lambda}_1$ on $\tilde{I}_t$. A useful property to give to $\tilde{I}_t$ in the discrete case can be to let the confidence interval beginning of $\tilde{\lambda}_1$ be increasing or decreasing. Such search intervals $\tilde{I}_t$ can be useful in the repeated calculations of action sets. In general it is even harder to design quick methods for the numerical calculation of action sets in the discrete case than it is in the continuous case.

5.2 A useful ordering of intervals

When designing some quick method to calculate numerically the action sets corresponding to states $(q, i-1)$ with $\tilde{B}_{i-1}(q)$ as (λ_0, λ) set, a subproblem arising usually is the following: Given a finite set A of intervals (of the form $[s_1, s_2]$) and a finite set B of intervals find to each interval $\tilde{I} \in B$ all intervals $I \in A$ for which

$$\tilde{I} \cap I \neq \emptyset. \tag{5.17}$$

If $I = [\underline{a}, \overline{a}]$, $\tilde{I} = [\underline{b}, \overline{b}]$ then the conditions

$$\underline{b} \leq \overline{a} \quad \text{and} \quad \underline{a} \leq \overline{b}$$

are necessary and sufficient conditions for (5.17). It may be good to first order the intervals of A in a way which makes it possible to find those intervals of A faster which satisfy (5.17) for any given $\tilde{I} \in B$. In the sequel an ordering of intervals in A will be shown with help of a little example to illustrate the method.

Example

Consider the following intervals:

$[1, 3], [2, 9], [3, 4], [2, 5], [3, 5]$

Order them in the following two dimensional way:

[1,3]	[2,5]	[2,9]	[3,5]	[3,4]
[1,3]	[2,5]	[2,9]	[2,9]	[2,9]
-	[1,3]	[2,5]	[3,5]	[3,5]
-	-	[1,3]	[2,5]	[2,5]
-	-	-	[1,3]	[3,4]
-	-	-	-	[1,3]

In the top row the intervals are ordered so that the interval beginnings are increasing. Below the horizontal line column i is the following: The first i intervals of the top row ordered so that the interval ends are decreasing from top to bottom. If now, for example, $\tilde{I} = [2, 3]$ and one wants to find all of the above five intervals which have a non-empty intersection with $\tilde{I}$ then one does the following: look in the top row for the interval which is as close to the right side as possible such that its interval beginning is $\leq$ the interval end of $\tilde{I}$, ie. 3.
This interval is [3,4], the last entry in the top row. Then stay in this column, move one interval down,(ie. to [2,9]), check whether this interval's end is $\geq$ interval beginning of $\tilde{I}$. If no, stop. If yes, then this interval has a non-empty intersection with $\tilde{I}$, move in the same column to the next interval down and repeat the checking on this interval. And so on....

Clearly, this leads directly to all intervals with non-empty intersection with $\tilde{I}$, once the correct interval in the top row has been found. But to find this top row entry only requires $O(\log k)$ operations using bisection search, k being the number of search intervals.

To establish the table requires the following work: $O(k \log k)$ operations to establish the top row using heap sort and $O(k^2)$ to establish the rest, since one can use column $i-1$ when establishing column i. All one has to do is to insert in column $i-1$ one more interval and shift the rest. So when establishing the columns under the horizontal line, working from left to right, only $O(k)$ operations for each column are needed.

5.3 Example with computational results

Consider the following problem of the form (5.1)-(5.3):

$$\max \sum_{h=1}^{r} f_h(x_h) \tag{5.18}$$

$$\text{subject to} \quad \sum_{h=1}^{r} x_h = \hat{q}, \tag{5.19}$$

$$x_h \in \mathbb{Z}^2, \quad 0 \leq x_{h,1} \leq 40 \qquad \forall h, \tag{5.20}$$

$$0 \leq x_{h,2} \leq 40 \qquad \forall h, \tag{5.21}$$

where $\hat{q} \in \mathbb{R}^2$ is given by $\hat{q}^T = (150, 70)$, $r = 6$ and the functions $f_n : \mathbb{Z}^2 \to \mathbb{R}$ are given by

$$f_n(x) = \begin{cases} \sqrt{x_1 + x_2 + n^2} & \text{if } x_2 \leq 25 \\ \sqrt{x_1 + x_2 + n^2} + \frac{1}{5}n(x_1 - x_2) & \text{if } x_2 > 25 \end{cases}$$

This problem has been implemented once with action elimination and once without. The results are as follows:

	run time (secs)	computed optimal objective
with action elimination	108.11	72.16
no action elimination	268.30	72.16

$$\text{speed-up achieved } = \frac{268.30}{108.11} = 2.48$$

The discretisation of the state space (q, i) was, of course, chosen to be $q \in \mathbb{Z}^2$, $0 \leq q_1 \leq 150$, $0 \leq q_2 \leq 70$ for all $i : 1 \leq i \leq 6$. As this problem has linear constraints with positive integer coefficients all states obtainable belong to this set of discretised states. If in (5.20) the right hand bound (*i.e.* 40) is replaced by a bigger number, for example by 60, then the ratio of run times of DP with and without action elimination changes in favour of the action elimination DP as can be seen from the following results:

	run time (secs)	computed optimal objective
with action elimination	134.41	114.04
no action elimination	423.85	114.04

$$\text{speed-up achieved } = \frac{423.85}{134.41} = 3.15$$

Conditions (5.9)-(5.12) are actually optimality conditions for problem (5.18)-(5.21). This will be proved next.

5.4 Derivation of a necessary local optimality condition

Theorem 5.4.1 *If in problem (5.1)- (5.3)* $w_h(x_h) = x_h \quad \forall h$ *then for all subproblems (5.5)-(5.7) there exists* (λ_0, λ) *with* $\lambda_0 = 1$, $\lambda \in \mathbb{R}^m$ *such that* (λ_0, λ) *and the optimal solution vector* $\widetilde{\boldsymbol{x}}_i$, *where* $\widetilde{\boldsymbol{x}}_i^T = (\tilde{x}_1^T, \tilde{x}_2^T, ..., \tilde{x}_i^T)$, *satisfy (5.9)-(5.12).*

Proof:

First of all notice that $w_h(x_h) = x_h \quad \forall h$ implies that all $n_h = m$. In order to

show that (5.9), or equivalently (5.13), holds for some (λ_0, λ) define the difference operators $\triangle_k^+$, $\triangle_k^-$ by

$$\begin{aligned} \triangle_k^+ g(x) &:= \max\{g(x+e_k) - g(x), \quad g(x) - g(x-e_k)\}, \\ \triangle_k^- g(x) &:= \min\{g(x+e_k) - g(x), \quad g(x) - g(x-e_k)\} \end{aligned}$$

where g is a function $\mathbb{Z}^n \to \mathbb{R}$ and e_k is again the k-th unit vector. When these operators are applied to f_h then redefine

$$\begin{aligned} \triangle_k^+ f_h(x_h) &:= +\infty \qquad \text{if } (x_h - e_k) \notin D_h, \\ \triangle_k^- f_h(x_h) &:= -\infty \qquad \text{if } (x_h + e_k) \notin D_h. \end{aligned}$$

This redefinition of the operators $\triangle_k^+, \triangle_k^-$ applied to f_h is useful in order to be in line with the definition of $\frac{\partial f_h}{\partial x_{h,k}}(x_h)$ given in (5.4). With the above definition of the operators $\triangle_k^+$, $\triangle_k^-$

$$\frac{\partial f_h}{\partial x_{h,k}}(x_h) = [\triangle_k^- f_h(x_h), \triangle_k^+ f_h(x_h)].$$

Let

$$a_k := \max_{h:1\le h\le i} \{\triangle_k^- f_h(\tilde{x}_h)\} \tag{5.22}$$

$$b_k := \min_{h:1\le h\le i} \{\triangle_k^+ f_h(\tilde{x}_h)\} \tag{5.23}$$

Suppose that $a_k > b_k$ for some k. Then let s be the argument of h which achieves the maximum in (5.22) and let t be the argument of h which achieves the minimum in (5.23). Then

$$\triangle_k^- f_s(\tilde{x}_s) > \triangle_k^+ f_t(\tilde{x}_t).$$

Since from the definition of $\triangle_k^-$, $\triangle_k^+$

$$\triangle_k^- f_h(\tilde{x}_h) \le \triangle_k^+ f_h(\tilde{x}_h) \qquad \forall h$$

it follows that $s \neq t$. Putting these arguments together: $a_k > b_k$ for some k implies that there exist indices s and t with $s \neq t$, $1 \le s \le i$, $1 \le t \le i$ such that

$$\triangle_k^+ f_s(\tilde{x}_s) \ge \triangle_k^- f_s(\tilde{x}_s) > \triangle_k^+ f_t(\tilde{x}_t) \ge \triangle_k^- f_t(\tilde{x}_t)$$

From this it follows that

$$f_s(\tilde{x}_s + e_k) - f_s(\tilde{x}_s) > f_t(\tilde{x}_t) - f_t(\tilde{x}_t - e_k).$$

Then the point $(\hat{x}_1, \hat{x}_2, .., \hat{x}_i)$ which is given by

$$\begin{aligned} \hat{x}_t &= \tilde{x}_t - e_k, \\ \hat{x}_s &= \tilde{x}_s + e_k, \\ \hat{x}_j &= \tilde{x}_j \qquad \text{for all } j \neq t, j \neq s, 1 \leq j \leq i \end{aligned}$$

satisfies

$$\sum_{h=1}^{i} \hat{x}_h = \sum_{h=1}^{i} \tilde{x}_h$$

and the objective value for $(\hat{x}_1, \hat{x}_2, .., \hat{x}_i)$ is

$$\begin{aligned} \sum_{h=1}^{i} f_h(\hat{x}_h) &= \sum_{h=1}^{i} f_h(\tilde{x}_h) + \Big(f_s(\tilde{x}_s + e_k) - f_s(\tilde{x}_s) \Big) \\ &\qquad + \Big(f_t(\tilde{x}_t - e_k) - f_t(\tilde{x}_t) \Big) \\ &> \sum_{h=1}^{i} f_h(\tilde{x}_h) \end{aligned}$$

and hence $\widetilde{\boldsymbol{x}}_i$ is not optimal, which contradicts the assumption. Therefore , $a_k \leq b_k$ for all k. Then there exist $\lambda_k \in I\!R$ such that

$$a_k \leq \lambda_k \leq b_k.$$

The definition of a_k and b_k implies

$$\begin{aligned} & \triangle_k^- f_h(\tilde{x}_h) \leq \lambda_k \leq \triangle_k^+ f_h(\tilde{x}_h) \quad \forall h \text{ and } \forall k \\ \Rightarrow \quad & \lambda_k \in \frac{\partial f_h}{\partial x_{h,k}}(\tilde{x}_h) \quad \forall h \text{ and } \forall k \end{aligned} \tag{5.24}$$

Since $w_h(x_h) = x_h$ it is the case that

$$\begin{aligned} \frac{\partial w_{h,j}}{\partial x_{h,k}}(\tilde{x}_h) &= [1,1] \quad \text{if } j = k \quad \text{and} \\ \frac{\partial w_{h,j}}{\partial x_{h,k}}(\tilde{x}_h) &= [0,0] \quad \text{if } j \neq k. \end{aligned}$$

Hence

$$\sum_{j=1}^{m} \lambda_j \frac{\partial w_{h,j}}{\partial x_{h,k}}(\tilde{x}_h) = [\lambda_k, \lambda_k]$$

This combined with (5.24) implies that (5.13) holds with $\lambda_0 = 1$. Remember that (5.13) is equivalent to (5.9). Hence there exists (λ_0, λ) such that $\widetilde{\boldsymbol{x}}_i$ and (λ_0, λ) satisfy (5.9)-(5.12). □

However, for general discrete problems of the form (5.1)-(5.3) it is not necessary that there exists (λ_0, λ) such that the optimal vector and (λ_0, λ) satisfy (5.9),(5.11), (5.12).

5.5 Counterexample

In this subsection a discrete DP problem is given for which there does not exist (λ_0, λ) such that the optimal vector and (λ_0, λ) satisfy (5.9),(5.11), (5.12). Consider the problem

$$\begin{aligned} &\text{maximize} && f_1(x_1) + f_2(x_2) && (5.25)\\ &\text{subject to} && 3x_1 + x_2 = 8, && (5.26)\\ & && x_1, x_2 \in \mathbb{Z}, \quad x_1, x_2 \geq 0, && (5.27) \end{aligned}$$

where the functions $f_1, f_2 : \mathbb{Z} \to \mathbb{R}$ are given by

x	0	1	2	3	4
$f_1(x)$	0	1	5	8	1
$f_2(x)$	0	1	3	5	1

and $f_1(x) = f_2(x) = 1$ for $x > 4$. As can be seen easily, the optimal vector to this problem is $(x_1, x_2) = (2, 2)$. However, for $(2, 2)$ condition (5.9), or equivalently (5.13), is:

$$\begin{aligned} &\lambda_0[3,4] \cap \lambda_1[3,3] \neq \emptyset \qquad \text{and}\\ &\lambda_0[2,2] \cap \lambda_1[1,1] \neq \emptyset. \end{aligned}$$

If $\lambda_0 = 0$ then it implies $\lambda_1 = 0$ which contradicts $(\lambda_0, \lambda_1) \neq 0$. If $\lambda_0 = 1$ then the second relation above implies $\lambda_1 = 2$ which contradicts the first relation above. However, if in problem (5.25)-(5.27) the RHS of (5.26), *i.e.* 8, is replaced with 7 then the optimal vector $(x_1, x_2) = (2, 1)$ does satisfy (5.9)-(5.9) for some (λ_0, λ_1) since (5.9) is

$$\begin{aligned} &\lambda_0[3,4] \cap \lambda_1[3,3] \neq \emptyset \qquad \text{and}\\ &\lambda_0[1,2] \cap \lambda_1[1,1] \neq \emptyset \end{aligned}$$

and this is satisfied, for example, for $(\lambda_0, \lambda_1) = (1, 1)$ or $(\lambda_0, \lambda_1) = (1, \frac{4}{3})$. Also if the RHS of (5.26), *i.e.* 8, is replaced with 9 then the optimal vector $(x_1, x_2) = (2, 3)$ satisfies (5.9)-(5.12) for some (λ_0, λ_1) (for example when $(\lambda_0, \lambda_1) = (1, 1)$).

For general problems of the form (5.1)-(5.3) if one looks at the subproblems corresponding to states (q, i), given by (5.5)-(5.7), it is found that for a lot of subproblems the optimal vector $\widetilde{\boldsymbol{x}}_i$ satisfies (5.9)-(5.12) for some (λ_0, λ), but for some subproblems the optimal vector $\widetilde{\boldsymbol{x}}_i$ does not satisfy (5.9)-(5.12) for any (λ_0, λ). The proportion of subproblems satisfying (5.9)-(5.12) for some (λ_0, λ) to those subproblems which do not satisfy (5.9)-(5.12) for any (λ_0, λ) depends on the functions w_h. Broadly speaking, the observation was made that if $\frac{\partial w_{h,j}}{\partial x_{h,k}}(x_h)$

for different h but fixed j, k are non-intersecting and far apart, then there are proportionately more subproblems (q, i) whose optimal vector $\boldsymbol{x}_i$ does not satisfy (5.9)-(5.12) for any (λ_0, λ).

5.6 Modified discrete action elimination algorithm

The counter-example of the last subsection is an example on which the discrete action elimination DP algorithm fails to discover the optimal solution. This example is so simple that this method does not seem to be appropriate for general problems of the form (5.1)-(5.3). However, the discrete action elimination method appears to be good when it is extended and combined with local search methods in the following way:

First, apply the discrete action elimination DP algorithm to problem (5.1)-(5.3) such that the value function $F_r(q)$ is computed for all discretised values of q in some neighbourhood of $\hat{q}$, *i.e.* for all q for which

$$|q - \hat{q}| \leq T. \tag{5.28}$$

T has to be estimated and chosen beforehand. Then, for every q satisfying (5.28) do a local search to problem (5.1)-(5.3) taking the computed optimal vector of the subproblem corresponding to state (q, r) as the starting point.

5.7 Conclusions

This chapter discussed action elimination based on local optimality conditions for discrete deterministic allocation problems of the form (5.1)-(5.3).

A special case is $w_h(x_h) = x_h$ and for this case the necessary local optimality conditions (5.9)-(5.12) were derived in Theorem 5.4.1. These conditions involve forward and backward differences and look similar to the FJ conditions for continuous problems. The action elimination presented, which is based on these local optimality conditions, always works in this case.

For the general case when $w_h(x_h) \neq x_h$ conditions (5.9)-(5.12) are not necessary optimality conditions. A counterexample was given, there DP with action elimination based on conditions (5.9)-(5.12) failed to find the optimum.

In section 5.6 a modified discrete action elimination algorithm was outlined. In this algorithm heuristics are applied. Some limited computational experimentation gave some encouraging results but more work has to be done in this area.

In particular it was found that it is hard to estimate an appropriate value for the parameter T in advance.

Chapter 6

Stochastic problems

In this chapter stochastic problems will be discussed and conditions will be formulated which allow action elimination based on first order optimality conditions. There are different classes of stochastic DP problems: infinite or finite horizon, continuous or discrete state and action space problems. Discrete time Markov decision processes are underlying all the problems discussed in this chapter.

6.1 Infinite horizon, continuous state and action space

The intention of this section is to introduce the basic idea underlying action elimination for stochastic problems. Let's first look at infinite horizon, continuous state and action space problems. Let $S = \mathbb{R}^n$ be the state space, let $A = \mathbb{R}^n$ be the action space. (Notice that S and A are unbounded and that A does not depend on $s \in S$. This is unrealistic but useful for simplifying the presentation of the basic idea.) Let $\mathcal{P}$ be a set of probability distributions on S with the properties below. In this subsection let $Y = \mathbb{R}^n$. (In the next subsection Y will be more specific.)

Property 6.1.1

a) For every $y \in Y$ there exists exactly one $P \in \mathcal{P}$ which is associated with y. Let $P(y)$ be the $P \in \mathcal{P}$ which is associated with y.

b) If $\mathbb{E}_P$ is the expectation operator for the probability distribution P then $\mathbb{E}_{P(y)} f$ is continuous and differentiable with respect to y, when f is any bounded function $S \to \mathbb{R}$.

c) If from state $s \in S$ the action $a \in A$ is taken then the transition to the next state is determined by the probability distribution $P(s - a)$.

Just for clarity of notation, if $p(s)$ is the probability density function of the distribution $P \in \mathcal{P}$ then

$$\mathbb{E}_P f = \int_S p(s) f(s) \mathrm{d}s$$

Let the instantaneous reward of taking action a from state s be $r(s,a)$ where $r(s,a)$ is a bounded function $S \times A \to \mathbb{R}$ and r is differentiable with respect to the argument a. Consider the following problem:

$$\text{maximize the expectation of } \sum_{j=0}^{\infty} \alpha^j r(s_j, a_j) \tag{6.1}$$

where α is a discount factor with $0 \leq \alpha < 1$, s_0 is the initial state of the system, $\{s_j\}_{j=0}^{\infty}$ is the Markov process induced by the sequence of actions $\{a_j\}_{j=0}^{\infty}$. This means the optimization is over the space of policies for taking actions. It is well known that for such infinite horizon, stationary Markov decision processes there exists an optimal policy π for taking actions which is Markovian, stationary and deterministic. *i.e.* π is a function $S \to A$ such that observing the system in period j to be in state s_j it is optimal to take action $a_j = \pi(s_j)$. (See Puterman [13] sections 4.4 and 5.5) Let $v(s)$ be the optimal objective value of problem (6.1) when the initial state $s_0 = s$. The DP optimality equation for problem (6.1) is:

$$v(s) = \max_{a \in A} \{r(s,a) + \alpha \mathbb{E}_{P(s-a)} v\}, \tag{6.2}$$

$$\pi(s) = \arg\max_{a \in A} \{r(s,a) + \alpha \mathbb{E}_{P(s-a)} v\} \tag{6.3}$$

In the sequel it is shown how to use action elimination based on first order optimality conditions for a value iteration step. Value iteration approximates $v(s)$ by $v_i(s)$ and $\pi(s)$ by $\pi_i(s)$ after iteration i. The iteration is as follows:

$$v_{i+1}(s) = \max_{a \in A} \{r(s,a) + \alpha \mathbb{E}_{P(s-a)} v_i\} \quad \forall s \in S \tag{6.4}$$

and the policy π_{i+1} of choosing actions after iteration $i+1$ is:

$$\pi_{i+1}(s) = \arg\max_{a \in A} \{r(s,a) + \alpha \mathbb{E}_{P(s-a)} v_i\} \quad \forall s \in S \tag{6.5}$$

A first order optimality condition for the maximization in the RHS of (6.4) is:

$$\nabla_a (r(s,a) + \alpha \mathbb{E}_{P(s-a)} v_i) = 0 \tag{6.6}$$

where ∇_a is the gradient operator with respect to $a \in \mathbb{R}^n$. (6.6) can be rewritten as

$$\nabla_a r(s,a) - \alpha \nabla_y (\mathbb{E}_{P(y)} v_i) = 0, \tag{6.7}$$

$$y = s - a \tag{6.8}$$

where ∇_y is the gradient operator with respect to $y \in I\!R^n$. (6.4) can be implemented using recursive fixing or the labelling procedure (also called reaching. See the beginning of section 2.3.) The recursive fixing implementation does not allow action elimination using (6.6), or equivalently (6.7), (6.8), whereas the labelling procedure does. The labelling procedure which implements (6.4), (6.5) without action elimination is:

for all $s \in S$ set $v_{i+1}(s) := -\infty$
for all $y \in Y$ do
  for all $a \in A$ do
    if $v_{i+1}(y+a) < r(y+a,a) + \alpha I\!E_{P(y)} v_i$ then
      $v_{i+1}(y+a) := r(y+a,a) + \alpha I\!E_{P(y)} v_i$
      $\pi_{i+1}(y+a) := a$
    end if
  end for
end for

The labelling procedure using (6.7), (6.8) for action elimination is as above except that line 3 is replaced by the following two lines:

$J := \{x \in A : \nabla_a r(y+x,x) = \alpha \nabla_y (I\!E_{P(y)} v_i)\}$
for all $a \in J$ do

This action elimination procedure shows the basic idea of action elimination for stochastic problems, which is based on first order optimality conditions. In this presentation the state space S and action space A are unbounded, which is not realistic for real world problems. And of course, the loops in the procedure above are infinite loops, and for any implementation a discretisation scheme must be used.

For bounded state and action spaces some modifications must be done which are similar to the modifications of section 2.4. In section 2.3 the basic action elimination procedure for problems of the form (2.1),(2.2) was introduced and the state and action spaces were unbounded. In section 2.4 the necessary modifications were shown when the action space in problem (2.1),(2.2) is restricted.

6.2 Extension to bounded state and action spaces

In the previous section the state space and action space were both $I\!R^n$, *i.e.* unbounded. In this section it is assumed that the state space $S \subset I\!R^n$ and that S

is bounded. Also it is assumed that the action space is restricted in the following way: A_s is the set of actions which can be taken from state s, $A_s \subset \mathbb{R}^n$ and A_s is bounded and closed. Taking action $a \in A_s$ from state s leads to a stochastic transition to a new state determined by the probability distribution $P(s-a)$ on S, $P(s-a) \in \mathcal{P}$. $\mathcal{P}$ satisfies Property 6.1.1. There exists a $P(y)$ for every $y \in Y$ where

$$Y := \{y \in \mathbb{R}^n : \exists s \in S, \exists a \in A_s \text{ such that } y = s - a\} \tag{6.9}$$

Let $r(s,a)$ be a piecewise differentiable function. The objective is, again, (6.1). Again, let $v(s)$ be the optimal objective value of (6.1) when the initial state $s_0 = s$, let $\pi(s)$ be the optimal action from A_s to be taken when the state of the system is s. The new DP optimality equations are similar to (6.2),(6.3):

$$v(s) = \max_{a \in A_s}\{r(s,a) + \alpha \mathbb{E}_{P(s-a)} v\}, \tag{6.10}$$

$$\pi(s) = \arg\max_{a \in A_s}\{r(s,a) + \alpha \mathbb{E}_{P(s-a)} v\} \tag{6.11}$$

For the value iteration let $v_i(s)$ and $\pi_i(s)$ be, again, the approximating functions for $v(s)$ and $\pi(s)$ respectively after iteration i. The iteration now is

$$v_{i+1}(s) = \max_{a \in A_s}\{r(s,a) + \alpha \mathbb{E}_{P(s-a)} v_i\} \quad \forall s \in S, \tag{6.12}$$

$$\pi_{i+1}(s) = \arg\max_{a \in A_s}\{r(s,a) + \alpha \mathbb{E}_{P(s-a)} v_i\} \quad \forall s \in S. \tag{6.13}$$

Let (6.14) be the following logical statement:

$$\nabla_a r \text{ does not exist at } (s,a) \tag{6.14}$$

When $s \in S$ and $a \in A_s$ is optimal for the maximization in the RHS of (6.12) then a necessary first order optimality condition is:

$$a \in \partial A_s \text{ or } [a \in A_s \backslash \partial A_s \text{ and } [(6.6) \text{ or } (6.14)]]$$

The brackets '[]' are used to indicate the order of the logical connectives 'and' and 'or'. The action elimination will use this condition. However, it is not easy to see how to use this condition within the labelling procedure. In order to see this let's replace (6.6) with the equivalent condition [(6.7) and (6.8)]. The above logical expression is equivalent to

$$a \in \partial A_s \text{ or } [a \in A_s \backslash \partial A_s \text{ and } [[(6.7) \text{ and } (6.8)] \text{ or } (6.14)]]$$

In the labelling procedure the outer loop cycles through all values $y \in Y$ and the inner loop cycles through values of a. The above condition makes it possible to

reduce the inner loop over a. In the above expression (6.8) can be used to replace all other occurrences of s with $y + a$. This might look a little bit artificial but it gives a motivation for the objects C_y and B_y which will be defined next and which will be used in the action elimination procedure.

$$C_y := \{a : \exists s \in S \text{ such that } a \in A_s \backslash \partial A_s, \quad y = s - a\} \tag{6.15}$$

In words: C_y is the set of actions a which, when taken from some state s, lead to a transition of state determined by the probability distribution $P(y)$ and a is in the interior of A_s.

$$B_y := \{a : \exists s \in S \text{ such that } a \in \partial A_s, \quad y = s - a\} \tag{6.16}$$

In words: B_y is the set of actions a which, when taken from some state s, lead to a transition of state determined by the probability distribution $P(y)$ and a is on the boundary of A_s.

Next, the value iteration step using the labelling procedure with action elimination will be shown.

Procedure 6.2.1

for all $s \in S$ set $v_{i+1}(s) := -\infty$
for all $y \in Y$ do
  $C_y := \{a : \exists s \in S \text{ such that } a \in A_s \backslash \partial A_s, \quad y = s - a\}$
  $B_y := \{a : \exists s \in S \text{ such that } a \in \partial A_s, \quad y = s - a\}$
  $J := \{x \in C_y : \nabla_a r(y + x, x) = \alpha \nabla_y (I\!E_{P(y)} v_i)$ or $\nabla_a r$ does not exist at $(y + x,$
  for all $a \in B_y \cup J$ do
    if $v_{i+1}(y + a) < r(y + a, a) + \alpha I\!E_{P(y)} v_i$ then
      $v_{i+1}(y + a) := r(y + a, a) + \alpha I\!E_{P(y)} v_i$
      $\pi_{i+1}(y + a) := a$
    end if
  end for
end for

This procedure is faster than an ordinary value iteration procedure without action elimination if the sets C_y, B_y can be found quickly from the structure of a problem and if J can be found quickly.

6.3 A special type of reward function

In this section a special type of reward function is looked at which allows a very simple implementation of action elimination within recursive fixing. The setting is as in the previous section 6.2 described there in the first paragraph.

Consider a reward function r of the following form:

$$r(s,a) = f(s) + g(s-a) \tag{6.17}$$

where f and g are functions $\mathbb{R}^n \to \mathbb{R}$. So f only depends on the present state and g, in some sense, depends only on the distribution determining the next state. The first order optimality condition (6.6) becomes:

$$\nabla_a(f(s) + g(s-a) + \alpha \mathbb{E}_{P(s-a)} v_i) = 0$$

This is equivalent to

$$\nabla_y(g(y) + \alpha \mathbb{E}_{P(y)} v_i) = 0, \tag{6.18}$$

$$y = s - a. \tag{6.19}$$

Notice that (6.18) does not depend on s and not on a. Now (6.14) is equivalent to

$$\nabla g(y) \text{ does not exist at } y = s - a \tag{6.20}$$

Taking action a from state s can only be optimal if

$$a \in \partial A_s \text{ or } [a \in A_s \backslash \partial A_s \text{ and } [(6.20) \text{ or } [(6.18) \text{ and } (6.19)]]] \tag{6.21}$$

This optimality condition can easily be implemented in the recursive fixing method as follows:

Procedure 6.3.1

$Q := \{y \in Y : \nabla g(y) + \alpha \nabla_y(\mathbb{E}_{P(y)} v_i) = 0$
or $\nabla g(y)$ does not exist$\}$
for all $s \in S$ do
  $v_{i+1}(s) := -\infty$
  $J := \{a \in A_s \backslash \partial A_s : \exists y \in Q$ such that $y = s - a\}$
  for all $a \in \partial A_s \cup J$ do
    if $v_{i+1}(s) < r(s,a) + \alpha \mathbb{E}_{P(s-a)} v_i$ then

$v_{i+1}(s) := r(s,a) + \alpha \mathbb{E}_{P(s-a)} v_i$

$\pi_{i+1}(s) := a$

end if

end for

end for

Note that if line 5 in above procedure is replaced with

$$\text{for all } a \in A_s \text{ do}$$

and lines 1 and 4 are deleted then this is the ordinary recursive fixing method, which is the most straightforward implementation for recursion (6.12).

Regarding the true solution $v(s)$ and $\pi(s)$ the following can be said. The RHS of (6.10) is like the RHS of (6.12) with the only difference that v_i is replaced with v. Hence, if in (6.21) all occurrences of v_i are replaced with v then this is a optimality condition for the optimal action $a = \pi(s)$ from state s. Therefore, if

$$Q := \{y \in Y : \nabla g(y) + \alpha \nabla_y (\mathbb{E}_{P(y)} v) = 0 \text{ or } \nabla g(y) \text{ does not exist}\}$$

then either

$\pi(s) \in \partial A_s$ or

$[\pi(s) \in A_s \backslash \partial A_s$ and $y = s - \pi(s)$ for some $y \in Q]$.

Broadly speaking, this says that the policy π tries to obtain a transition of state corresponding to a set of preferred probability distributions on S.

6.4 Example

In this section an example from water reservoir management is discussed to illustrate the theory of the previous two sections.

Consider a water reservoir for which the state is determined by the water level. *i.e.* $n = 1$ for this problem. If the reservoir is empty then the state is 0. The maximal height of the water level is H, because water flows over a dam if too much water is in the reservoir. This means that the state space can be characterised by

$$S = [0, H]$$

At day j in the morning water can be released from the reservoir. If from state s_j action a_j is taken, which corresponds to releasing the amount of water which

decreases the water level by a_j in height, then the new water level is $s_j - a_j$. Of course $0 \leq a_j \leq s_j$. This means

$$A_s = [0, s] \quad \forall s \in S.$$

The release of water happens instantaneously and yields a reward $r(s_j, a_j)$. r is a bounded function. During the next 24 hours fresh water from streams flows into the reservoir, it rains and some water from the reservoir evaporates. Let $\{X_i\}_{i=0}^{\infty}$ be a sequence of independent, identically distributed random variables with normal distribution $\mathrm{N}(\mu, \sigma^2)$. The state s_{i+1} of the reservoir the following morning, before water is released, is given by

$$s_{i+1} = \begin{cases} 0 & \text{if } s_i - a_i + X_i \leq 0, \\ H & \text{if } s_i - a_i + X_i \geq H, \\ s_i - a_i + X_i & \text{otherwise.} \end{cases} \tag{6.22}$$

When the initial state is s_0 then the long term expected maximal discounted revenue is given by (6.1) with $0 \leq \alpha < 1$. The transition rule (6.22) is satisfied if $P(y)$ is a normal distribution $\mathrm{N}(y + \mu, \sigma^2)$ and the domain of the approximating functions v_i is extended for $s \notin S$ by

$$\begin{aligned} v_i(s) &:= v_i(0) \qquad \text{for } s < 0, \\ v_i(s) &:= v_i(H) \qquad \text{for } s > H. \end{aligned}$$

With this extension of v_i and with $y = s_i - a_i$ it is the case that

$$\mathbb{E}v_i(s_{i+1}) = \mathbb{E}_{P(y)}v_i = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu-y)^2}{2\sigma^2}} v_i(x)\mathrm{d}x$$

and the fact that $\mathrm{N}(\mu + y, \sigma^2)$ is not a probability distribution on S but on $(-\infty, \infty)$ is taken care of. In order to apply Procedure 6.2.1 for this water reservoir problem Y, C_y, B_y must be found. Using (6.9),(6.15),(6.16) one finds

$$\begin{aligned} &Y = [0, H], \\ &C_0 = C_H = \emptyset, \\ &C_y = (0, H - y] \quad \text{for } y \in (0, H), \\ &B_0 = [0, H], \\ &B_y = \{0\} \quad \text{for } y \in (0, H]. \end{aligned}$$

A practical procedure for a value iteration step with action elimination for this problem is the following procedure which is based on Procedure 6.2.1. If Procedure 6.2.1 is applied to this problem directly then line 2 is the start of a loop $y \in [0, H]$. The procedure below is obtained by splitting this loop into two loops

$y \in \{0\}$ and $y \in (0, H]$. Line 1 in Procedure 6.2.1 and the execution of the loop $y \in \{0\}$ first is equivalent to line 2 below.

for all $y \in Y$ compute $I\!E_{P(y)} v_i$
for all $s \in S$ set $v_{i+1}(s) := r(s,s) + \alpha I\!E_{P(0)} v_i$ and $\pi_{i+1}(s) := s$
for all $y \in (0, H]$ do
  $J := \{x \in C_y : \frac{\partial}{\partial a} r(y+x, x) = \alpha \frac{\mathrm{d}}{\mathrm{d}y}(I\!E_{P(y)} v_i)$
    or $\frac{\partial}{\partial a} r$ does not exist at $(y+x, x)\}$
  for all $a \in J \cup \{0\}$ do
    if $v_{i+1}(y+a) < r(y+a, a) + \alpha I\!E_{P(y)} v_i$ then
      $v_{i+1}(y+a) := r(y+a, a) + \alpha I\!E_{P(y)} v_i$
      $d_{i+1}(y+a) := a$
    end if
  end for
end for

So far the reward function r in this problem has not been specified. Assume that the reward $r(s,a)$ comes from hydroelectric power generation. Let the potential energy stored in the water reservoir with water level s be $g(s)$. Decreasing the water level by a from a water level s the energy produced is $g(s) - g(s-a)$, hence

$$r(s,a) = K(g(s) - g(s-a))$$

for some constant K. This reward function falls under the special case (6.17), therefore section 6.3 can be applied, *i.e.* Procedure 6.3.1 can be used instead of Procedure 6.2.1.

If on top of revenue from hydroelectric power generation the released water is also sold and if the reservoir has vertical shores and a flat ground then the amount of water released is proportional to the action a. This means

$$\begin{aligned} r(s,a) &= K_1(g(s) - g(s-a)) + K_2 a \\ &= K_1 g(s) + K_2 s - K_1 g(s-a) - K_2(s-a) \end{aligned}$$

where K_1, K_2 are constants. Again, r is of the special form (6.17).

6.5 Action elimination when $\partial A_s = A_s$

In this section the action elimination procedure is discussed for problems where $\partial A_s \approx A_s$. Notice that if $\partial A_s = A_s$ for all $s \in S$ then Procedure 6.2.1 is in

essence a labelling procedure with no action elimination taking place. Often for problems with $\partial A_s \approx A_s$ and special structure action elimination can still be used. The key idea is that even for optimal actions $a \in \partial A_s$ there often exist necessary optimality conditions which can be used for action elimination. Let's look at an example to see what is meant.

Let $S \subset \mathbb{R}^2$ be given by $S = [0, H_1] \times [0, H_2]$. Let A_s for $s \in S$ be given as

$$A_s = \{(a_1, a_2) \in \mathbb{R}^2 : -s_1 \leq a_1 \leq H_1 - s_1, -s_2 \leq a_2 \leq H_2 - s_2, a_1 a_2 = 0\}$$

where $s = (s_1, s_2)$. Taking action $a \in A_s$ from state s leads to a stochastic transition to a new state determined by the probability distribution $P(s - a)$ on S. From (6.9)

$$Y = S = [0, H_1] \times [0, H_2]$$

A necessary optimality condition for the maximization in the RHS of (6.12) is

$$a \in \{(-s_1, 0), (H_1 - s_1, 0), (0, -s_2), (0, H_2 - s_2)\}$$

$$\text{or} \quad \frac{\partial}{\partial a_1}(r(s,a) + \alpha \mathbb{E}_{P(s-a)} v_i) = 0 \tag{6.23}$$

$$\text{or} \quad \frac{\partial}{\partial a_2}(r(s,a) + \alpha \mathbb{E}_{P(s-a)} v_i) = 0 \tag{6.24}$$

where a_1, a_2 are the first and second components of $a \in \mathbb{R}^2$. When setting $y = s - a$ (6.23) and (6.24) are

$$\frac{\partial}{\partial a_1} r(y + a, a) - \alpha \frac{\partial}{\partial y_1} \mathbb{E}_{P(y)} v_i = 0,$$
$$\frac{\partial}{\partial a_2} r(y + a, a) - \alpha \frac{\partial}{\partial y_2} \mathbb{E}_{P(y)} v_i = 0$$

where y_1, y_2 are the first and second components of $y \in \mathbb{R}^2$. These conditions can be used for action elimination within the labelling procedure.

6.6 Infinite horizon, discrete state and action space

The material and framework in this section is similar to that of section 6.2 but there are important differences. It is assumed that the state space $S \subset \mathbb{Z}^n$ and the action space $A_s \subset \mathbb{Z}^n$. Taking action $a \in A_s$ from state s leads to a stochastic transition to a new state determined by the probability distribution $P(s - a)$ on S. Let Y be the set

$$Y := \{y \in \mathbb{Z}^n : \exists s \in S, \exists a \in A_s, y = s - a\}$$

Since actions are discrete the meaning of interior and of boundary of A_s must be given. Let e_k be the k-th unit vector in $\mathbb{Z}^n$ *i.e.* the vector with the k-th component equal to 1 and the other components equal to 0. The boundary of A_s is

$$\partial A_s := \{a \in A_s : \exists k \text{ with } 1 \leq k \leq n \text{ such that } (a + e_k) \notin A_s \text{ or } (a - e_k) \notin A_s\}$$

The interior is $A_s \backslash \partial A_s$. Again, let $v(s)$ be the optimal objective value of (6.1) when the initial state $s_0 = s$, let $\pi(s)$ be the optimal action from A_s to be taken when the state of the system is s. The DP optimality equations are (6.10) and (6.11). The value iteration is (6.12) and (6.13) where $v_i(s), \pi_i(s)$ are the approximating functions for $v(s)$, $\pi(s)$ respectively after iteration i. If a is optimal for the maximization in the RHS of (6.12) then there are two possibilities: the first is $a \in \partial A_s$, the second is

$$a \in A_s \backslash \partial A_s,$$

$$r(s, a + e_k) + \alpha I\!E_{P(s-a-e_k)} v_i \leq r(s, a) + \alpha I\!E_{P(s-a)} v_i \quad \forall k, \tag{6.25}$$

$$r(s, a - e_k) + \alpha I\!E_{P(s-a+e_k)} v_i \leq r(s, a) + \alpha I\!E_{P(s-a)} v_i \quad \forall k. \tag{6.26}$$

Define the difference operators $\triangle_k^+$, $\triangle_k^-$ operating on the functions $r(s, a)$ and $I\!E_{P(y)} f$ by

$$\begin{aligned}
\triangle_k^+ r(s, a) &:= r(s, a + e_k) - r(s, a), \\
\triangle_k^- r(s, a) &:= r(s, a) - r(s, a - e_k), \\
\triangle_k^+ I\!E_{P(y)} f &:= I\!E_{P(y+e_k)} f - I\!E_{P(y)} f, \\
\triangle_k^- I\!E_{P(y)} f &:= I\!E_{P(y)} f - I\!E_{P(y-e_k)} f.
\end{aligned}$$

Notice that

$$\begin{aligned}
\triangle_k^- r(s, a) &= \triangle_k^+ r(s, a - e_k), \\
\triangle_k^- I\!E_{P(y)} f &= \triangle_k^+ I\!E_{P(y-e_k)} f.
\end{aligned}$$

With these definitions (6.25) is equivalent to

$$\triangle_k^+ r(s, a) \leq \alpha \triangle_k^- I\!E_{P(s-a)} v_i \quad \forall k$$

and (6.26) is equivalent to

$$\triangle_k^- r(s, a) \geq \alpha \triangle_k^+ I\!E_{P(s-a)} v_i \quad \forall k.$$

After setting $y = s - a$

$$\Delta_k^+ r(y+a,a) \leq \alpha \Delta_k^- I\!E_{P(y)} v_i \quad \forall k, \tag{6.27}$$

$$\Delta_k^- r(y+a,a) \geq \alpha \Delta_k^+ I\!E_{P(y)} v_i \quad \forall k. \tag{6.28}$$

Let C_y and B_y be given by (6.15) and (6.16) respectively. Next, the value iteration step using the labelling procedure with action elimination will be shown. This procedure is identical to Procedure 6.2.1 except of line 5 (which defines the set J). In the next section the procedure below is referred to and used for an example which is why the procedure will be given in full.

Procedure 6.6.1

for all $s \in S$ set $v_{i+1}(s) := -\infty$
for all $y \in Y$ do
  $C_y := \{a : \exists s \in S \text{ such that } a \in A_s \backslash \partial A_s, \quad y = s - a\}$
  $B_y := \{a : \exists s \in S \text{ such that } a \in \partial A_s, \quad y = s - a\}$
  $J := \{x \in C_y : (6.27) \text{ and } (6.28) \text{ hold}\}$
  for all $a \in B_y \cup J$ do
    if $v_{i+1}(y+a) < r(y+a,a) + \alpha I\!E_{P(y)} v_i$ then
      $v_{i+1}(y+a) := r(y+a,a) + \alpha I\!E_{P(y)} v_i$
      $\pi_{i+1}(y+a) := a$
    end if
  end for
end for

Like for continuous action and state space problems, if the reward function has the form

$$r(s,a) = f(s) + g(s-a)$$

then it is possible (and easier) to implement the action elimination with recursive fixing. Then (6.27) and (6.28) reduce to

$$g(y - e_k) - g(y) \leq \alpha \Delta_k^- I\!E_{P(y)} v_i \quad \forall k, \tag{6.29}$$

$$g(y) - g(y + e_k) \geq \alpha \Delta_k^+ I\!E_{P(y)} v_i \quad \forall k. \tag{6.30}$$

One can use Procedure 6.3.1 with line 1 being replaced with

$$Q := \{y \in Y : (6.29) \text{ and } (6.30) \text{ hold}\}.$$

6.7 Example with computational results

In this section an example problem is given to illustrate the theory of the last section. Consider the problem where there are $n+1$ places numbered from 0 to n. The aim is to reach place 0 with minimal cost (or maximal negative cost) from place n. When one is currently at a place with number s one can decide how much to jump closer to place 0. However, deciding to jump a places from place s towards 0 results in a stochastic transition to a new place which only depends on $s-a$. The cost of each jump depends on the current place s and on how far one wants to jump *i.e.* on a. This is the rough description of the problem, the exact mathematical description of the problem follows next.

Let the state space S be given as

$$S = \{s \in \mathbb{Z} : 0 \leq s \leq n\}$$

where n determines the size of the problem. Let the sets A_s be given by

$$A_s = \{a \in \mathbb{Z} : 0 \leq a \leq s\}.$$

This means that

$$\begin{aligned} \partial A_s &= \{0, s\}, \\ Y &= \{y \in Z : 0 \leq y \leq n\} \qquad (= S). \end{aligned}$$

Let the probability distribution $P(y)$ be characterised by the probability density function $p_y(s)$. State zero is an absorbing state, this means

$$p_0(s) = \begin{cases} 1 & \text{if } s = 0 \\ 0 & \text{otherwise} \end{cases}$$

When $0 < y < n$ let

$$p_y(s) = \begin{cases} \frac{1}{2} & \text{if } s = y \\ \frac{1}{4} & \text{if } s = y \pm 1 \\ 0 & \text{otherwise} \end{cases}$$

and

$$p_n(s) = \begin{cases} \frac{3}{4} & \text{if } s = n \\ \frac{1}{4} & \text{if } s = n-1 \\ 0 & \text{otherwise} \end{cases}$$

So when f is a function $S \to \mathbb{R}$ then

$$\mathbb{E}_{P(y)} f = \sum_{s=0}^{n} p_y(s) f(s)$$

n	run time Method 1 (secs)	run time Method 2 (secs)	result
20	4.00E-02	5.00E-02	-56.33
50	0.16	9.99E-02	-154.47
100	0.67	0.29	-331.08
200	2.35	0.75	-698.09
500	18.18	2.21	-1749.91
1000	94.71	7.33	-3055.04
2000	491.26	15.72	-4504.54

Table 6.1: computational results

Let $r(s,a)$ be given by

$$r(s,a) = -(s+1)^{\frac{1}{2}} - a^{\frac{3}{2}}$$

Let the objective be (6.1) with $s_0 = n$ and $\alpha = 0.99$. Let this problem be solved by the Modified Policy Iteration Algorithm. Method 1 does the policy improvement step using recursive fixing with no action elimination. Method 2 does the policy improvement step using Procedure 6.6.1. Methods 1 and 2 are identical regarding the initial choice of $v_0(s)$ and parameters which must be set in the Modified Policy Iteration Algorithm. Since recursive fixing and Procedure 6.6.1 are equivalent Method 1 and Method 2 compute the same result. For different choices of n, which determines the size of the problem, the results are shown in Table 6.1. These results suggest that for this problem the complexity of Method 1 is one order of magnitude larger than that of Method 2.

Next, some details of Procedure 6.6.1 applied to this problem are discussed. In line 3 of Procedure 6.6.1 the sets C_y for $y \in Y$ are

$$\begin{aligned} C_0 &= C_n = \emptyset, \\ C_y &= \{a \in \mathbb{Z} : 0 < a \leq n - y\} \quad \text{for } 0 < y < n. \end{aligned}$$

In line 4 the sets B_y are

$$\begin{aligned} B_0 &= \{a \in \mathbb{Z} : 0 \leq a \leq n\}, \\ B_y &= \{0\} \quad \text{for } 0 < y \leq n. \end{aligned}$$

Crucial for the speed-up of Procedure 6.6.1 in comparison to recursive fixing with no action elimination is that the set J can be found quickly. When $y = 0$ or $y = n$ then $J = \emptyset$ since $C_y = \emptyset$ in this case. When $0 < y < n$ then J can be found faster than by checking all $a \in C_y$ whether (6.27) and (6.28) hold. In (6.27) and (6.28) only $k = 1$ is possible since $S \subset \mathbb{Z}$ and $A_s \subset \mathbb{Z}$.

$$\triangle_1^+ r(y+a,a) = -(a+1)^{\frac{3}{2}} + a^{\frac{3}{2}}$$

$\triangle_1^+ r(y+a,a)$ is strictly decreasing in a since

$$\frac{\mathrm{d}}{\mathrm{d}a}(-(a+1)^{\frac{3}{2}} + a^{\frac{3}{2}}) = \frac{3}{2}(a^{\frac{1}{2}} - (a+1)^{\frac{1}{2}} < 0 \quad \text{when } a \geq 0$$

Further

$$\triangle_1^- r(y+a,a) = \triangle_1^+ r(y+a-1, a-1)$$

since $\triangle_1^- r(y+a,a) = -a^{\frac{3}{2}} + (a-1)^{\frac{3}{2}}$. Therefore

$$\triangle_1^+ r(y+a,a) < \triangle_1^- r(y+a,a). \tag{6.31}$$

Now, since $\frac{\partial}{\partial a} r(y+a,a) = -\frac{3}{2}a^{\frac{1}{2}}$ is monotonically decreasing and $\triangle_1^+ r(y+a,a)$ is a difference approximation to $\frac{\partial}{\partial a} r(y+a,a)$ the following relation holds:

$$\triangle_1^- r(y+a,a) \geq \frac{\partial}{\partial a} r(y+x,x) \geq \triangle_1^+ r(y+a+1, a+1) \tag{6.32}$$

when $a \in S$ and $x \in \mathbb{R}$ with $a \leq x \leq a+1$.

$$\frac{\partial}{\partial a} r(y+x,x) = \lambda$$

can be solved analytically for x. This and relation (6.32) can be used to find the smallest $a \in C_y$ for which (6.27) is fulfilled. By checking (6.28), then increasing a by one and checking (6.28) and so on until (6.28) fails for the first time, the set J can be found quickly.

6.8 Finite horizon problems

Consider the problem

$$\text{maximize } \mathbb{E}\left(\sum_{j=0}^{N-1} r_j(s_j, a_j) \quad + r_N(s_N)\right) \tag{6.33}$$

where $\{s_j\}_{j=0}^N$ is the Markov process induced by the sequence of actions $\{a_j\}_{j=0}^{N-1}$ corresponding to some decision policy. So the optimization is over the set of all decision policies. s_0 is the initial state of the process. The state space is S and the set of actions possible to take from state $s \in S$ is A_s. The action space is $A = \bigcup_{s \in S} A_s$. The functions r_j are functions $S \times A \to \mathbb{R}$ for $j < N$, r_N is a function $S \to \mathbb{R}$. r_N is the terminal reward. Taking action a_j from state s_j leads to a stochastic transition to a new state s_{j+1} determined by the probability distribution $P_{j+1}(s_j - a_j)$ on S. Define

$$v_0(s) := r_N(s) \qquad \forall s \in S$$

and

$$v_i(s) = \max \mathbb{E}\left(\sum_{j=N-i}^{N-1} r_j(s_j, a_j) + r_N(s_N) \middle| s_{N-i} = s\right) \qquad \forall s \in S$$

when $i > 0$. So the optimal solution of (6.33) has the objective value $v_N(s_0)$. The DP optimality equation can be derived using conditional expectations and it is

$$v_{i+1}(s) = \max_{a \in A_s} \left\{ r_{N-i-1}(s,a) + I\!E_{P_{N-i}(s-a)} v_i \right\} \quad \forall s \in S.$$

This is similar to recursion (6.12). Hence, for doing the above recursion one can use the ideas and methods of sections 6.2, 6.3 if $S \subset I\!R^n$ and $A_s \subset I\!R^n$ or of section 6.6 if $S \subset \mathbb{Z}^n$ and $A_s \subset \mathbb{Z}^n$.

6.9 Conclusions

This chapter discussed action elimination based on local optimality conditions for stochastic problems. The ideas were similar as in previous chapters, however, the appropriate framework had to be set up. For stochastic problems a lot of structure in the problem is required in order to be able to apply the action elimination. Many real world problems have this structure. Further classes of stochastic DP problems allowing action elimination based on first order optimality conditions can probably be found.

For a class of infinite horizon problems with continuous state and action space and for a class of infinite horizon problems with discrete state and action space the details of action elimination based on local optimality conditions were shown. For classes of finite horizon problems it was shown how previous ideas can be applied.

The computational results of a discrete problem showed that the action elimination can reduce the computation time by one order of magnitude.

Chapter 7

A perturbation method in a dual algorithm

7.1 The dual method

Consider the following problem:

$$\begin{aligned} &\text{maximize} && f(x) && (7.1)\\ &\text{subject to} && u(x) \leq 0, && (7.2)\\ & && c(x) = 0 && (7.3)\\ & && x \in X && (7.4) \end{aligned}$$

where $X \subset I\!R^n$, the functions f, u, c are $f : I\!R^n \to I\!R$, $u : I\!R^n \to I\!R^k$, and $c : I\!R^n \to I\!R^m$. Define the Lagrangian function

$$L(\lambda, \mu) := \max_{x \in X} \{f(x) - \lambda^T u(x) - \mu^T c(x)\} \tag{7.5}$$

where $\lambda \in I\!R^k$, $\mu \in I\!R^m$. This means that the constraints (7.2) and (7.3) are relaxed. $L(\lambda, \mu)$ is a convex function. This will be proved next.

Lemma 7.1.1 *$L(\lambda, \mu)$ is convex in λ and μ.*

Proof:
Let $0 \leq \phi \leq 1$ and $(\lambda_1, \mu_1) \neq (\lambda_2, \mu_2)$ where $\lambda_1, \lambda_2 \in I\!R^k$ and $\mu_1, \mu_2 \in I\!R^m$. Then

$$\begin{aligned} & L(\phi\lambda_1 + (1-\phi)\lambda_2, \phi\mu_1 + (1-\phi)\mu_2) \\ = & \max_{x \in X}\{f(x) \quad - (\phi\lambda_1^T + (1-\phi)\lambda_2^T)u(x) \quad - (\phi\mu_1^T + (1-\phi)\mu_2^T)c(x)\} \\ = & \max_{x \in X}\{\phi(f(x) - \lambda_1^T u(x) - \mu_1^T c(x)) \quad + (1-\phi)(f(x) - \lambda_2^T u(x) - \mu_2^T c(x))\} \\ \leq & \phi \max_{x \in X}\{f(x) - \lambda_1^T u(x) - \mu_1^T c(x)\} \quad + (1-\phi) \max_{x \in X}\{f(x) - \lambda_2^T u(x) - \mu_2^T c(x)\} \\ = & \phi L(\lambda_1, \mu_1) \quad + (1-\phi) L(\lambda_2, \mu_2) \qquad \square. \end{aligned}$$

Suppose that the problem (7.1)-(7.4) is such that there exists a method with which $L(\lambda, \mu)$ can be computed for every choice of λ, μ. The dual problem of problem (7.1)-(7.4) is

$$\begin{aligned} \text{minimize} \quad & L(\lambda, \mu) && (7.6) \\ \text{subject to} \quad & \lambda \geq 0. && (7.7) \end{aligned}$$

Lemma 7.1.2 *If $\lambda \geq 0$ then $L(\lambda, \mu)$ is an upper bound for the optimal objective value of problem (7.1)-(7.4).*

Proof:
Suppose x is feasible for problem (7.1)-(7.4). By definition of $L(\lambda, \mu)$

$$L(\lambda, \mu) \geq f(x) - \lambda^T u(x) - \mu^T c(x) = f(x) - \lambda^T u(x)$$

Since $\lambda \geq 0$ and $u(x) \leq 0$ it follows that $\lambda^T u(x) \leq 0$ and hence $L(\lambda, \mu) \geq f(x)$ $\square$.

This means that the dual problem (7.6), (7.7) is the minimization of upper bounds on the optimal objective value of problem (7.1)-(7.4). Let $\hat{\lambda}, \hat{\mu}$ be a solution to the above dual problem. The dual method first solves the dual problem, and then tries to find a "good" point $\hat{x}$ for the original problem (7.1)-(7.4) by picking $\hat{x}$ from the set S where

$$S := \{x : x \in X, f(x) - \hat{\lambda}^T u(x) - \hat{\mu}^T c(x) = L(\hat{\lambda}, \hat{\mu})\}.$$

If $\hat{x}$ satisfies constraints (7.2) and (7.3) and is complementary slack (ie. $\hat{\lambda}^T u(\hat{x}) = 0$) then it solves the original problem. This will be proved below. However, there is no guarantee that $\hat{x}$ satisfies (7.2),(7.3) and the complementary slackness condition. Often the set S has more than one element.

Theorem 7.1.3 (Lagrange's Sufficiency Theorem) *Suppose there exists λ, μ and $\hat{x}$ such that*

$$\begin{aligned} \hat{x} &\in \arg\max_{x \in X} \{f(x) - \lambda^T u(x) - \mu^T c(x)\}, \\ c(\hat{x}) &= 0, \\ u(\hat{x}) &\leq 0, \\ \lambda &\geq 0, \\ \lambda^T u(\hat{x}) &= 0. \end{aligned}$$

Then $\hat{x}$ solves problem (7.1)-(7.4).

Proof:

By Lemma 7.1.2 $L(\lambda, \mu)$ is an upper bound for the optimal objective value of problem (7.1)-(7.4). From the properties of $\hat{x}$ it follows that $\hat{x}$ is feasible for problem (7.1)-(7.4) and that

$$L(\lambda, \mu) = f(\hat{x}) - \lambda^T u(\hat{x}) - \mu^T c(\hat{x}) \quad = f(\hat{x}) \qquad \square.$$

The next Lemma shows the effect of changing a Lagrange multiplier. As the previous results of this section it has been known for a long time.

Lemma 7.1.4 *Let $\lambda_2 = \lambda_1 + \epsilon e_i$ where e_i is the $i-th$ unit vector and $\epsilon \in \mathbb{R}$ with $\epsilon > 0$. Let x_1, x_2 satisfy*

$$x_1 \in \arg\max_{x \in X} \{f(x) - \lambda_1^T u(x) - \mu^T c(x)\}, \tag{7.8}$$

$$x_2 \in \arg\max_{x \in X} \{f(x) - \lambda_2^T u(x) - \mu^T c(x)\}. \tag{7.9}$$

Then $u_i(x_2) \leq u_i(x_1)$, where $u_i(x)$ is the i-th component of $u(x)$.

Proof:

$$f(x_1) - \mu^T c(x_1) - \lambda_2^T u(x_1) = f(x_1) - \mu^T c(x_1) - \lambda_1^T u(x_1) - (\lambda_2 - \lambda_1)^T u(x_1) \tag{7.10}$$

$$f(x_2) - \mu^T c(x_2) - \lambda_2^T u(x_2) = f(x_2) - \mu^T c(x_2) - \lambda_1^T u(x_2) - (\lambda_2 - \lambda_1)^T u(x_2) \tag{7.11}$$

From (7.9) it follows that LHS of (7.10) $\leq$ LHS of (7.11). This implies

$$\begin{aligned} & f(x_1) - \mu^T c(x_1) - \lambda_1^T u(x_1) - (\lambda_2 - \lambda_1)^T u(x_1) \\ \leq\ & f(x_2) - \mu^T c(x_2) - \lambda_1^T u(x_2) - (\lambda_2 - \lambda_1)^T u(x_2) \\ \Leftrightarrow\quad & f(x_1) - \mu^T c(x_1) - \lambda_1^T u(x_1) \quad - (f(x_2) - \mu^T c(x_2) - \lambda_1^T u(x_2)) \\ \leq\ & (\lambda_2 - \lambda_1)^T (u(x_1) - u(x_2)) \end{aligned} \tag{7.12}$$

From (7.8) it follows that 0 $\leq$LHS of (7.12).

$$\begin{aligned} \Rightarrow \quad 0 &\leq (\lambda_2 - \lambda_1)^T (u(x_1) - u(x_2)) \\ \Rightarrow \quad 0 &\leq \epsilon (u_i(x_1) - u_i(x_2)) \\ \Rightarrow \quad u_i(x_2) &\leq u_i(x_1) \qquad \square. \end{aligned}$$

Corollary 7.1.5 *Let $\mu_2 = \mu_1 + \epsilon e_i$ where e_i is the $i-th$ unit vector and $\epsilon > 0$. Let x_1, x_2 satisfy*

$$x_1 \in \arg\max_{x \in X} \{f(x) - \lambda^T u(x) - \mu_1^T c(x)\},$$

$$x_2 \in \arg\max_{x \in X} \{f(x) - \lambda^T u(x) - \mu_2^T c(x)\}.$$

Then $c_i(x_2) \leq c_i(x_1)$.

Proof: It is similar to the proof of Lemma 7.1.4. □.

The next theorem exhibits a subtangent from the subdifferential set $\partial L(\lambda, \mu)$.

Theorem 7.1.6 *Consider the subdifferential set $\partial L(\hat{\lambda}, \hat{\mu})$ and let $\hat{x}$ satisfy*

$$\hat{x} \in \arg\max_{x \in X} \{f(x) - \hat{\lambda}^T u(x) - \hat{\mu}^T c(x)\}.$$

Let $p = \begin{pmatrix} -u(\hat{x}) \\ -c(\hat{x}) \end{pmatrix}$. *Then p is a subtangent vector of $\partial L(\hat{\lambda}, \hat{\mu})$, i.e. $p \in \partial L(\hat{\lambda}, \hat{\mu})$, i.e.*

$$L(\lambda, \mu) \geq L(\hat{\lambda}, \hat{\mu}) + p^T \begin{pmatrix} \lambda - \hat{\lambda} \\ \mu - \hat{\mu} \end{pmatrix} \tag{7.13}$$

for all $\lambda \in \mathbb{R}^k$, $\mu \in \mathbb{R}^m$.

Proof:
By definition of $L(\lambda, \mu)$

$$L(\lambda, \mu) \geq f(\hat{x}) - \lambda^T u(\hat{x}) - \mu^T c(\hat{x}) \tag{7.14}$$

Now, (7.13) and (7.14) are equivalent because (7.13) $\Leftrightarrow$

$$\begin{aligned} & L(\lambda, \mu) \geq L(\hat{\lambda}, \hat{\mu}) - u(\hat{x})^T(\lambda - \hat{\lambda}) - c(\hat{x})^T(\mu - \hat{\mu}) \\ \Leftrightarrow \quad & L(\lambda, \mu) \geq f(\hat{x}) - \hat{\lambda}^T u(\hat{x}) - \hat{\mu}^T c(\hat{x}) - u(\hat{x})^T(\lambda - \hat{\lambda}) - c(\hat{x})^T(\mu - \hat{\mu}) \\ \Leftrightarrow \quad & L(\lambda, \mu) \geq f(\hat{x}) - \lambda^T u(\hat{x}) - \mu^T c(\hat{x}). \end{aligned}$$

The last line is identical to (7.14). □.

7.1.1 Example

Consider the following problem:

$$\begin{aligned} \text{maximize} \quad & f_1(x_1) + f_2(x_2) \\ \text{subject to} \quad & s_1(x_1) + s_2(x_2) - 1.5 \leq 0, \\ & x_1 + x_2 = 1, \\ & x_1, x_2 \geq 0 \end{aligned} \tag{7.15}$$

where the functions f_1, f_2, s_1, s_2 are given as

$$f_1(x) = \begin{cases} 0 & \text{if } x < 1 \\ 3 & \text{if } x \geq 1 \end{cases}$$

$$f_2(x) = \begin{cases} 0 & \text{if } x < 1 \\ 2 & \text{if } x \geq 1 \end{cases}$$

$$s_1(x) = \begin{cases} 0 & \text{if } x < 1 \\ 2 & \text{if } x \geq 1 \end{cases}$$

$$s_2(x) = \begin{cases} 0 & \text{if } x < 1 \\ 1 & \text{if } x \geq 1 \end{cases}$$

It is clear from inspection that this problem has the unique solution $(x_1, x_2) = (0, 1)$. With the notation of (7.1)- (7.4) let X be the set $X = \{(x_1, x_2) : x_1, x_2 \geq 0, x_1 + x_2 = 1\}$. Then for this problem the dual function $L(\lambda)$ is defined by

$$L(\lambda) = \max_{\{x_1, x_2\}} \{f_1(x_1) + f_2(x_2) - \lambda(s_1(x_1) + s_2(x_2) - 1.5) \\ : x_1 + x_2 = 1, x_1 \geq 0, x_2 \geq 0\} \tag{7.16}$$

In this simple example $L(\lambda)$ can be found explicitly for every λ.

$$L(\lambda) = \begin{cases} 3 - 0.5\lambda & \text{if } \lambda \leq 1 \\ 2 + 0.5\lambda & \text{if } \lambda > 1 \end{cases}$$

The dual problem is

$$\begin{array}{ll} \text{minimize} & L(\lambda) \\ \text{subject to} & \lambda \geq 0 \end{array}$$

The solution to the dual problem is $\lambda = 1$. However, for $\lambda = 1$ there are two arguments (x_1, x_2) which achieve the maximum in the RHS of (7.16), namely $(1, 0)$ and $(0, 1)$. Of these $(0, 1)$ is the optimum to problem (7.15) whereas $(1, 0)$ is an infeasible point. For $\lambda < 1$ and $\lambda > 1$ the arguments (x_1, x_2) which achieve the maximum in the RHS of (7.16) are $(1, 0)$ and $(0, 1)$ respectively. It is worth mentioning that the solution to the dual problem $\lambda = 1$ and the solution to the primal problem $(x_1, x_2) = (0, 1)$ are not complementary slack.

If the evaluation of $L(\lambda)$ in (7.16) were done using dynamic programming and if the dual problem were solved numerically with a local optimization algorithm, then the dual method applied to problem (7.15) could return as result either $(x_1, x_2) = (1, 0)$ or $(x_1, x_2) = (0, 1)$.

7.2 Visualising the dual problem

In this section a problem of the form (7.1)- (7.4) with only one constraint being relaxed is looked at. Consider the problem

$$\begin{array}{lrcl} \text{maximize } f(x) & & & \\ \text{subject to} & c(x) & = & 0, \\ & x & \in & X, \end{array} \tag{7.17}$$

where $X \in \mathbb{R}^n$, f and c are functions $\mathbb{R}^n \to \mathbb{R}$. Define the following new function:

$$\overline{f}(\overline{c}) := \max_{x \in X} \{f(x) : c(x) = \overline{c}\}. \tag{7.18}$$

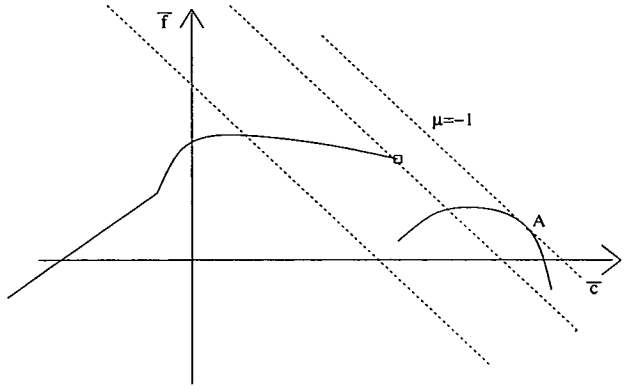

Figure 7.1:

i.e. $\overline{f}$ is a function $\mathbb{R} \to \mathbb{R}$. If $\bar{c}$ is such that there exists no $x \in X$ with $c(x) = \bar{c}$ then define $\overline{f}(\bar{c}) := -\infty$. Note that $\overline{f}(0)$ is the optimal objective value of problem (7.17). Also let the Lagrangian be

$$L(\mu) := \max_{x \in X} \{f(x) - \mu c(x)\} \tag{7.19}$$

where $\mu \in \mathbb{R}$. The dual problem is

$$\text{minimize} \quad L(\mu). \tag{7.20}$$

Assume, again, that there is a method with which for every $\mu \in \mathbb{R}$ it is possible to determine $L(\mu)$ and $x(\mu)$ where $x(\mu)$ is the argument achieving the maximum in the RHS of (7.19).

In the sequel different shapes of the function $\overline{f}$ will be discussed. In Figure 7.1 several features are worth mentioning. The function $\overline{f}(\bar{c})$ can be non smooth and discontinuous since the functions $f(x)$ and $c(x)$ are general non-linear functions. Even when $f(x)$ and $c(x)$ are continuous it is possible that $\overline{f}(\bar{c})$ is discontinuous. From definitions (7.18) and (7.19) it follows that

$$\begin{aligned} L(\mu) &= \max_{x \in X} \{f(x) - \mu c(x)\} \\ &= \max_{\bar{c}} \max_{x \in X} \{f(x) - \mu c(x) : c(x) = \bar{c}\} \\ &= \max_{\bar{c}} \{\overline{f}(\bar{c}) - \mu \bar{c}\}. \end{aligned}$$

In Figure 7.1 the evaluation of $L(\mu)$ for $\mu = -1$ is considered. $\overline{f} - \mu\bar{c}$ is constant on straight lines with slope μ. On two different straight lines with slope μ the value $\overline{f} - \mu\bar{c}$ is bigger on the line which lies above the other. Hence $L(\mu)$ can be associated with the straight line $\overline{f} - \mu\bar{c} = K$ (where K is a constant) which intersects the graph $\overline{f}(\bar{c})$ and which lies above all other straight lines with slope μ intersecting the graph $\overline{f}(\bar{c})$. The straight line associated with $L(\mu)$ intersects the

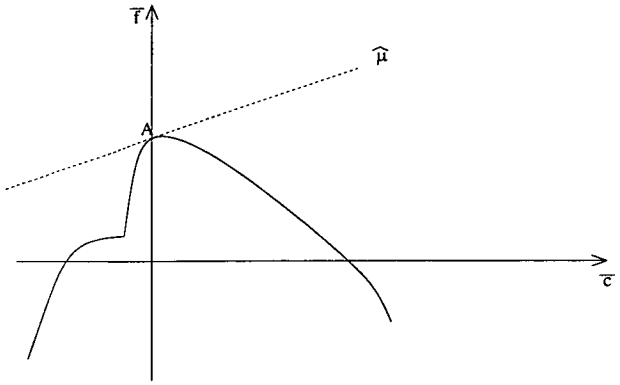

Figure 7.2:

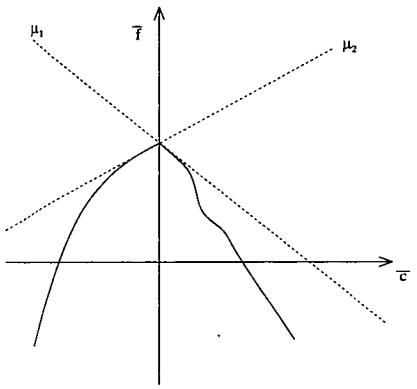

Figure 7.3:

$\overline{f}$-axis in the point $(0, L(\mu))$ and the graph $\overline{f}(\overline{c})$ in the point A. Let $\hat{x}$ be an argument which achieves the maximum in the RHS of (7.19). Then the coordinates of the point A in Figure 7.1 are $(c(\hat{x}), f(\hat{x}))$.

In Figure 7.2 the curve $\overline{f}(\overline{c})$ is smooth at $\overline{c} = 0$ and the function $L(\mu)$ is minimized for one value of μ, $\mu = \hat{\mu}$. The straight line with slope $\hat{\mu}$ associated with $L(\hat{\mu})$ intersects the graph $\overline{f}(\overline{c})$ at the point A with coordinates $(0, \overline{f}(0))$. When

$$\hat{x} \in \arg\max_{x \in X} \{f(x) - \hat{\mu} c(x)\} \tag{7.21}$$

then $c(\hat{x}) = 0$ and $f(\hat{x}) = L(\hat{\mu})$. This means that $\hat{x}$ solves problem (7.17), since every $L(\mu)$ is an upper bound on the optimal objective value of problem (7.17). (This follows from Lemma 7.1.2.)

In Figure 7.3 the graph $\overline{f}(\overline{c})$ is non-smooth at $\overline{c} = 0$. μ_1 and μ_2 are the slopes of the two straight lines. In Figure 7.3 all $\hat{\mu}$ with $\mu_1 \leq \hat{\mu} \leq \mu_2$ solve the dual problem (7.20).

In the example of Figure 7.4 $L(\mu)$ is minimized by $\hat{\mu}$. The straight line associated with $L(\hat{\mu})$ intersects the graph $\overline{f}(\overline{c})$ at two points A_1 and A_2. This means

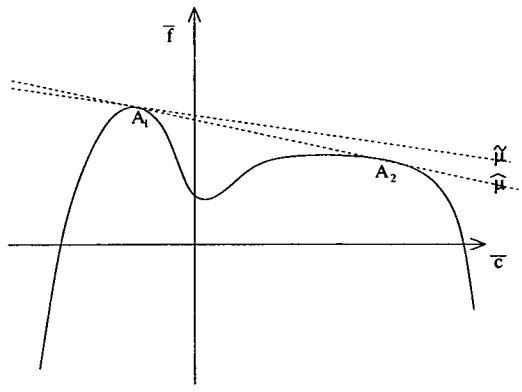

Figure 7.4:

that there exist $\hat{x}_1$ and $\hat{x}_2$ such that

$$\hat{x}_i \in \arg\max_{x \in X}\{f(x) - \hat{\mu}c(x)\} \quad \text{for } i = 1, 2 \tag{7.22}$$

and $(c(\hat{x}_1), f(\hat{x}_1))$ are the coordinates of point A_1 and $(c(\hat{x}_2), f(\hat{x}_2))$ are those of A_2. Note that $c(\hat{x}_1) \neq 0$, $c(\hat{x}_2) \neq 0$. So $\hat{x}_1$ and $\hat{x}_2$ are solutions with a duality gap. Further, since $(0, \overline{f}(0))$ lies in a valley of the graph $\overline{f}(\overline{c})$ there exists no μ such that the straight line associated with $L(\mu)$ intersects the graph $\overline{f}(\overline{c})$ at $(0, \overline{f}(0))$. More formally: there exist no μ and $\hat{x}$ such that

$$\begin{aligned} \hat{x} \quad &\in \quad \arg\max_{x \in X}\{f(x) - \mu c(x)\} \\ \text{and} \quad & \quad c(\hat{x}) = 0. \end{aligned}$$

If $L(\mu)$ is evaluated with a method which also returns as result exactly one $x(\mu)$ with

$$x(\mu) \in \arg\max_{x \in X}\{f(x) - \mu c(x)\} \tag{7.23}$$

then in the example of Figure 7.4 it is pure luck which of $\hat{x}_1, \hat{x}_2$ is returned as $x(\hat{\mu})$. This remark is important because in later sections Dynamic Programming will be used to evaluate a Lagrangian function and exactly one optimal argument x will be returned for every Lagrangian evaluation. In Figure 7.4 the point A_1 is closer to the $\overline{f}$-axis than A_2. Therefore $\hat{x}_1$ is a better approximation to the solution of problem (7.17) than $\hat{x}_2$, where $\hat{x}_1$, $\hat{x}_2$ satisfy (7.22) and give rise to the coordinates of A_1 and A_2 respectively. In order to find the argument $\hat{x}$ satisfying (7.21) and being nearest to the $\overline{f}$-axis (*i.e.* with $|c(\hat{x})|$ being smallest) the following perturbation method can be applied:

When $\hat{\mu}$ minimizes $L(\mu)$ and a Lagrangian evaluation method returns $x(\hat{\mu})$ as the optimal argument of the Lagrangian evaluation, set $\tilde{\mu} = \hat{\mu} + \epsilon c(x(\hat{\mu}))$ where $\epsilon > 0$ and ϵ is small. Evaluate $L(\tilde{\mu})$ and look at $c(x(\tilde{\mu}))$. If

$$|c(x(\tilde{\mu}))| < |c(x(\hat{\mu}))|$$

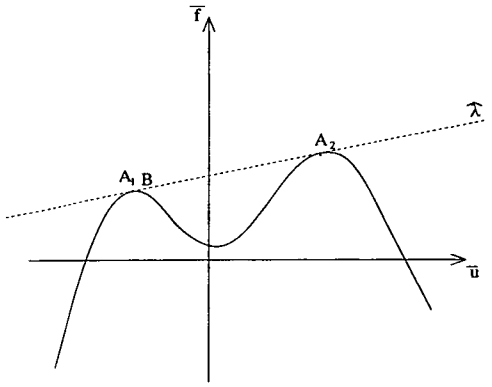

Figure 7.5:

then accept $x(\tilde{\mu})$ as the optimal argument of the Lagrangian evaluation of $L(\hat{\mu})$. Note that if $c(x(\hat{\mu})) > 0$ then

$$c(x(\tilde{\mu})) \leq c(x(\hat{\mu}))$$

by Corollary 7.1.5. If $c(x(\hat{\mu})) < 0$ then $c(x(\tilde{\mu})) \geq c(x(\hat{\mu}))$ by the same Corollary. In some sense this perturbation method assures that both sides of the valley in Figure 7.4 are checked.

Now let's look at an inequality constraint problem.

$$\begin{aligned} &\text{maximize } f(x) \\ &\text{subject to} \quad u(x) \leq 0, \\ &\qquad\qquad\quad x \in X, \end{aligned} \tag{7.24}$$

where $X \in I\!R^n$, f and u are functions $I\!R^n \to I\!R$. Define the equivalent to (7.18) and (7.19):

$$\overline{f}(\overline{u}) := \max_{x \in X} \{f(x) : u(x) = \overline{u}\}.$$

If $\overline{u}$ is such that there exists no $x \in X$ with $u(x) = \overline{u}$ then define $\overline{f}(\overline{u}) := -\infty$. The Lagrangian is

$$L(\lambda) := \max_{x \in X} \{f(x) - \lambda u(x)\}$$

where $\lambda \in I\!R$. The dual problem is

$$\begin{aligned} &\text{minimize} \quad L(\lambda) \\ &\text{subject to } \lambda \geq 0. \end{aligned}$$

In Figure 7.5 the point B corresponds to the optimal $\tilde{x}$ solving problem (7.24), *i.e.* B has coordinates $(u(\tilde{x}), f(\tilde{x}))$. The points A_1, A_2 are those corresponding to

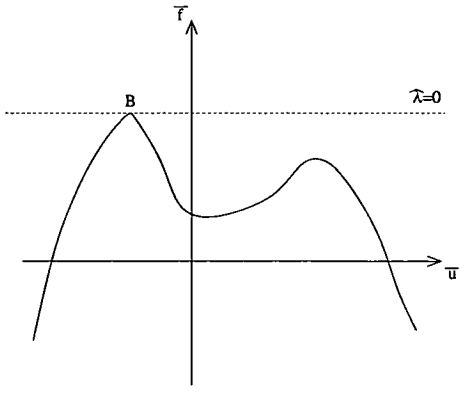

Figure 7.6:

optimal arguments $\hat{x}$ in the Lagrangian evaluation of $L(\hat{\lambda})$ where $\hat{\lambda}$ solves the dual problem. A_1, A_2 and the corresponding arguments $\hat{x}_1, \hat{x}_2$ are found by perturbing $\hat{\lambda}$ as in the equality constraint problem discussed before.

In Figure 7.6 B is again the point corresponding to the optimal $\tilde{x}$ solving problem (7.24), *i.e.* B has coordinates $(u(\tilde{x}), f(\tilde{x}))$. Since the dual problem has the constraint $\lambda \geq 0$ the dual method finds B and $\tilde{x}$ in the example of Figure 7.6.

7.3 Perturbation method in general

In the previous section it was described how to find the points A_1, A_2 in the example of Figure 7.4 by perturbing $\hat{\mu}$, where $\hat{\mu}$ is the solution point of the dual minimization problem (7.20). To be able to do this is important when $(0, \overline{f}(0))$ lies in a valley of the graph $\overline{f}(\overline{c})$ and when there is an evaluation method for $L(\mu)$ which returns only one optimal argument $x(\mu)$ satisfying (7.23) but not the whole set

$$\arg\max_{x \in X}\{f(x) - \mu c(x)\}.$$

However, the illustrations of the previous section used examples with one dimensional constraint space. It is necessary to generalise the perturbation method for problems with two or more constraints.

Let's look at problems of the form (7.1)-(7.4). The definition of $\overline{f}$ is (a generalisation of (7.18)):

$$\overline{f}(\overline{u}, \overline{c}) := \max_{x \in X}\{f(x) : u(x) = \overline{u}, c(x) = \overline{c}\} \tag{7.25}$$

where now $\overline{u} \in I\!R^k$, $\overline{c} \in I\!R^m$ and $\overline{f}$ is a function $I\!R^{k+m} \to I\!R$. If there is no $x \in X$ such that $u(x) = \overline{u}$ and $c(x) = \overline{c}$ then define $\overline{f}(\overline{u}, \overline{c}) := -\infty$. The Lagrangian $L(\lambda, \mu)$ is given by (7.5) where $\lambda \in I\!R^k$, $\mu \in I\!R^m$. The dual problem is given by

(7.6) and (7.7). From (7.5) and (7.25) the following relation can be derived:

$$\begin{aligned} L(\lambda,\mu) &= \max_{x\in X}\{f(x)-\lambda^T u(x)-\mu^T c(x)\} \\ &= \max_{\overline{u},\overline{c}}\max_{x\in X}\{f(x)-\lambda^T u(x)-\mu^T c(x) : u(x)=\overline{u}, c(x)=\overline{c}\} \\ &= \max_{\overline{u},\overline{c}}\{\overline{f}(\overline{u},\overline{c})-\lambda^T\overline{u}-\mu^T\overline{c}\} \end{aligned}$$

Assume that there is a method which for every $\lambda \in \mathbb{R}^k$, $\mu \in \mathbb{R}^m$ determines $L(\lambda,\mu)$ and a

$$x(\lambda,\mu) \in \arg\max_{x\in X}\{f(x)-\lambda^T u(x)-\mu^T c(x)\} \tag{7.26}$$

as corresponding optimal argument. Let $(\hat{\lambda},\hat{\mu})$ be a solution point of the dual problem (7.6),(7.7). Let S be the set of all optimal arguments $x \in X$ of the maximization problem corresponding to the evaluation of $L(\hat{\lambda},\hat{\mu})$. i.e.

$$S := \arg\max_{x\in X}\{f(x)-\hat{\lambda}^T u(x)-\hat{\mu}^T c(x)\} \tag{7.27}$$

The aim is now to obtain a $\tilde{x} \in S$ such that $||u(\tilde{x})_+||$ and $||c(\tilde{x})||$ are small, where u_+ is the vector u with all negative components replaced by zero. $||u(\tilde{x})_+||$ and $||c(\tilde{x})||$ being small means that the violations of constraints (7.2),(7.3) of the point $\tilde{x}$ are small. Analogous to the examples in Figures 7.2-7.6 the following can be said about a multi-constrained problem of the form (7.1)-(7.4).

Consider the space $\mathbb{R}^{k+m+1}$ where the first k axes correspond to components of $\overline{u}$, the next m axes correspond to components of $\overline{c}$ and the $m+k+1$-th axis corresponds to $\overline{f}$. In this space the hyperplane given by

$$\overline{f}-\hat{\lambda}^T\overline{u}-\hat{\mu}^T\overline{c} = L(\hat{\lambda},\hat{\mu}), \tag{7.28}$$

where $(\hat{\lambda},\hat{\mu})$ is a solution point of the dual problem, touches the graph $\overline{f}(\overline{u},\overline{c})$ given by (7.25) at at least one point. Furthermore, if constraints (7.2)-(7.4) are feasible then the hyperplane lies above the graph $\overline{f}(\overline{u},\overline{c})$ in the direction of the $\overline{f}$-axis, *i.e.*

$$L(\hat{\lambda},\hat{\mu})+\hat{\lambda}^T\overline{u}+\hat{\mu}^T\overline{c} \geq \overline{f}(\overline{u},\overline{c}) \qquad \forall \overline{u},\overline{c}. \tag{7.29}$$

If constraints (7.2)-(7.4) are feasible and if there exists no point A with coordinates $(u(x),c(x),f(x))$ for some $x \in X$ which lies on the hyperplane (7.28) and the graph $\overline{f}(\overline{u},\overline{c})$ and for which $u(x) \leq 0$ and $c(x)=0$ then the hyperplane touches the graph $\overline{f}(\overline{u},\overline{c})$ at more than one point. In this case a perturbation method is necessary in order to find a $\tilde{x} \in S$ with $||u(\tilde{x})_+||$ and $||c(\tilde{x})||$ small. The difficulty is how to choose perturbation directions such that in consecutive perturbations

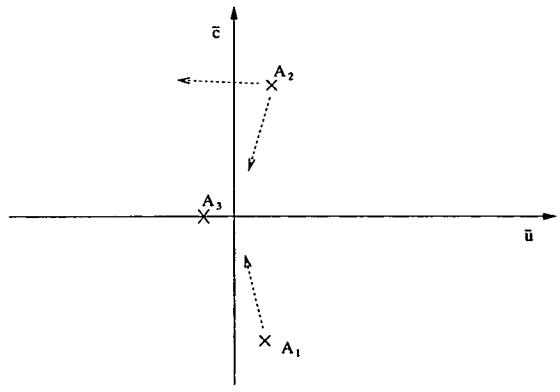

Figure 7.7:

no $x \in S$ is discovered twice and yet the whole space of relevant perturbation directions is covered. This point will be illustrated by the following example.

Consider a problem of the form (7.1)-(7.4) with $k = 1$ and $m = 1$. Figure 7.7 shows the $(\overline{u}, \overline{c}, \overline{f})$ space projected onto the $(\overline{u}, \overline{c})$ space where A_1, A_2, A_3 are three points in the $(\overline{u}, \overline{c}, \overline{f})$ space where the hyperplane given by (7.28) touches the graph $\overline{f}(\overline{u}, \overline{c})$. This means that at A_1, A_2, A_3 (7.29) holds as an equality. Suppose that an evaluation method for $L(\hat{\lambda}, \hat{\mu})$ returns the optimal argument $x(\hat{\lambda}, \hat{\mu}) \in S$ which corresponds to the point A_1, *i.e.* the coordinates of A_1 are $(u(x(\hat{\lambda}, \hat{\mu})), c(x(\hat{\lambda}, \hat{\mu})), f(x(\hat{\lambda}, \hat{\mu})))$. Among A_1, A_2, A_3 only the point A_3 corresponds to a primal feasible $x \in S$. The normal vector of hyperplane (7.28) is

$$\begin{pmatrix} -\hat{\lambda} \\ -\hat{\mu} \\ 1 \end{pmatrix}.$$

The perturbation of this vector in direction

$$\begin{pmatrix} -u(x(\hat{\lambda}, \hat{\mu})) \\ -c(x(\hat{\lambda}, \hat{\mu})) \\ 0 \end{pmatrix} \tag{7.30}$$

corresponds to setting

$$\begin{aligned} \lambda &= \hat{\lambda} + \epsilon u(x(\hat{\lambda}, \hat{\mu})) \\ \mu &= \hat{\mu} + \epsilon c(x(\hat{\lambda}, \hat{\mu})) \end{aligned}$$

for some small $\epsilon > 0$. The hyperplane with normal vector equal to this perturbed normal vector which touches $\overline{f}(\overline{u}, \overline{c})$ and lies above $\overline{f}(\overline{u}, \overline{c})$ in direction of the $\overline{f}$-axis is given by

$$\overline{f} - \lambda^T \overline{u} - \mu^T \overline{c} = L(\lambda, \mu).$$

This hyperplane touches $\overline{f}(\overline{u},\overline{c})$ in a point close to A_2 since among A_1,A_2,A_3 the point A_2 lies furthest away from the origin in the direction of perturbation (7.30). This means that the point $x(\lambda,\mu)$ returned by the evaluation method for $L(\lambda,\mu)$ will give rise to the coordinates of a point close to A_2 in the $(\overline{u},\overline{c},\overline{f})$ space. Now, having found A_1 and A_2, if in a similar way the normal vector of hyperplane (7.28) is perturbed in direction

$$\begin{pmatrix} -u(x(\lambda,\mu)) \\ -c(x(\lambda,\mu)) \\ 0 \end{pmatrix} \tag{7.31}$$

then a point close to A_1 will be found since among A_1,A_2,A_3 the point A_1 lies furthest away from the origin in the direction of perturbation (7.31). The point A_1 had been found before, therefore the perturbation direction (7.31) is not good. A better second perturbation vector is one which is orthogonal to the vector $(\vec{A_2} - \vec{A_1})$ and whose scalar product with $\vec{A_1}$ and $\vec{A_2}$ is negative as shown in Figure 7.7. Such a perturbation vector ensures that among A_1, A_2, A_3 none of the already found points A_1, A_2 is furthest away from the origin in the perturbation direction, hence neither A_1 nor A_2 will be found next.

The following is an algorithmic description of the perturbation process:

Procedure 7.3.1

1) Let $x_0 = x(\hat{\lambda},\hat{\mu})$ where $(\hat{\lambda},\hat{\mu})$ is an optimal point of the dual problem (7.6),(7.7). Let

$$a_0 = \begin{pmatrix} u(x_0) \\ c(x_0) \end{pmatrix},$$

i.e. a_0 is a $k+m$ dimensional vector.
If $u(x_0) \leq 0$, $c(x_0) = 0$ and $\hat{\lambda}^T u(x_0) = 0$ then set $i = 0$ and go to step 8.

2) There are l indices j such that $\hat{\lambda}_j = 0$ (for some $l \geq 0$). For each of such indices j include the $k+m$ dimensional unit vector with 1 as j-th component in the set B, *i.e.* B is a set of $k+m$ dimensional unit vectors and B has l elements. (If $l = 0$ then $B = \emptyset$)

3) set $i = 0$ and $D_0 = \emptyset$.

4) Pick a $k+m$ dimensional perturbation vector $p_{i+1} \neq 0$ which satisfies

$$\begin{aligned} a_i^T p_{i+1} &< 0, \\ v^T p_{i+1} &= 0 \qquad \text{for all } v \in D_i, \\ v^T p_{i+1} &\leq 0 \qquad \text{for all } v \in B. \end{aligned}$$

If no such vector exists go to step 8.

5) Pick a small $\epsilon > 0$ and set

$$\begin{aligned} \begin{pmatrix} \lambda \\ \mu \end{pmatrix} &= \begin{pmatrix} \hat{\lambda} \\ \hat{\mu} \end{pmatrix} - \epsilon p_{i+1}, \\ x_{i+1} &= x(\lambda, \mu), \\ a_{i+1} &= \begin{pmatrix} u(x_{i+1}) \\ c(x_{i+1}) \end{pmatrix}. \end{aligned}$$

If $u(x_i) \leq 0$, $c(x_i) = 0$ and $\hat{\lambda}^T u(x_i) = 0$ then increase i by one and go to step 8.

6) Set $D_{i+1} = D_i \cup \{a_{i+1} - a_0\}$.

7) Increase i by one and return to step 4.

8) Let E be an ordered set with elements x_0, x_1,... x_i. The order of elements in E is given by the magnitude of $||u(x)_+|| + ||c(x)||$, *i.e.* if $\tilde{x}, x^* \in E$ and $\tilde{x} < x^*$ then

$$||u(\tilde{x})_+|| + ||c(\tilde{x})|| \leq ||u(x^*)_+|| + ||c(x^*)||.$$

Stop.

Step 1 defines the first argument $x_0 \in S$ which gives rise to the coordinates of a point in the $(\overline{u}, \overline{c}, \overline{f})$ space for which (7.29) holds as equality. a_0 is the coordinate vector of this point projected onto the $(\overline{u}, \overline{c})$ space. If $u(x_0) \leq 0$, $c(x_0) = 0$ and $\hat{\lambda}^T u(x_0) = 0$ then x_0 is a global optimum by Lagrange's Sufficiency Theorem (Theorem 7.1.3). In this case there is no need to carry on with the procedure, therefore the procedure goes to step 8 and then terminates.

Step 2 deals with those components of $\hat{\lambda}$ which are on the boundary of the dual feasibility region. In these components perturbation can only take place in one direction in order to avoid the perturbed vector $\begin{pmatrix} \lambda \\ \mu \end{pmatrix}$ being dual infeasible by having a negative component in λ. By defining the set B in step 2 and by requiring in step 4 that p_{i+1} satisfies $v^T p_{i+1} \leq 0$ for all $v \in B$ it is achieved that the perturbed vector $\begin{pmatrix} \lambda \\ \mu \end{pmatrix}$ is dual feasible.

Step 4 defines a new perturbation direction p_{i+1}. The two conditions imposed on p_{i+1}, that $a_i^T p_{i+1} < 0$ and $v^T p_{i+1} = 0$ for all $v \in D_i$, imply that $a_k^T p_{i+1} < 0$ for all $0 \leq k \leq i$ since D_i contains all vectors of the form $a_k - a_0$ for $1 \leq k \leq i$. The formal derivation of this is:

Lemma 7.3.2 *If* $a_i^T p_{i+1} < 0$ *and* $(a_k^T - a_0^T) p_{i+1} = 0$ *for all* $0 \leq k \leq i$ *then* $a_k^T p_{i+1} < 0$ *for all* $0 \leq k \leq i$.

Proof:

$$\begin{aligned} a_k^T p_{i+1} &= a_k^T p_{i+1} + (a_i^T - a_0^T) p_{i+1} \\ &= (a_k^T - a_0^T) p_{i+1} + a_i^T p_{i+1} \\ &= a_i^T p_{i+1} \\ &< 0. \qquad \square. \end{aligned}$$

Step 5 does the perturbation of $\begin{pmatrix} \hat{\lambda} \\ \hat{\mu} \end{pmatrix}$ in direction $-p_{i+1}$, *i.e.* the normal vector $\begin{pmatrix} -\hat{\lambda} \\ -\hat{\mu} \\ 1 \end{pmatrix}$ of hyperplane (7.28) is perturbed in direction $\begin{pmatrix} p_{i+1} \\ 0 \end{pmatrix}$, and gets x_{i+1} which approximates a $\tilde{x} \in S$. $\begin{pmatrix} u(x_{i+1}) \\ c(x_{i+1}) \\ f(x_{i+1}) \end{pmatrix}$ is a point of the $(\overline{u}, \overline{c}, \overline{f})$ space approximating the point $\begin{pmatrix} u(\tilde{x}) \\ c(\tilde{x}) \\ f(\tilde{x}) \end{pmatrix}$ for which (7.29) holds as equality and which is furthest away from the origin in the direction of perturbation $\begin{pmatrix} p_{i+1} \\ 0 \end{pmatrix}$. a_{i+1} is the point $\begin{pmatrix} u(x_{i+1}) \\ c(x_{i+1}) \\ f(x_{i+1}) \end{pmatrix}$ projected onto the $(\overline{u}, \overline{c})$ space. At the end of step 5 there is a stopping criterion which is identical to the one at the end of step 1.

Step 6 defines D_{i+1} as the set of vectors which the next perturbation direction must be orthogonal to. Note that for $i \geq 1$

$$D_i = \{a_1 - a_0, a_2 - a_0, ..., a_i - a_0\}.$$

After this procedure one picks the point among x_0,x_1,..., x_i which least violates the constraints (7.2), (7.3), *i.e.* for which $||u(x)_+|| + ||c(x)||$ is least, and accepts it as an approximation to the solution point of problem (7.1)-(7.4). This will be the first element of E, where E is defined in step 8.

At this stage the set E does not seem to be of any use. Later on, Algorithm 8.4.1 will use Procedure 7.3.1 and there the set E will be useful.

Procedure 7.3.1 is always guaranteed to terminate. The reason for this is that the vectors

$$a_1 - a_0, a_2 - a_0, ..., a_i - a_0$$

are linearly independent (this will be proved in Theorem 7.3.4). These vectors are $k + m$ dimensional vectors. This implies that $i \leq k + m$. When $i = k + m$ then in step 4 there can not exist a vector $p_{i+1} \neq 0$ satisfying $p_{i+1}^T v = 0$ for all

$v \in D_i$ because

$$\{a_1 - a_0, a_2 - a_0, ..., a_i - a_0\} = D_i.$$

This leads to termination of Procedure 7.3.1.

Next the proof will be provided that $a_1 - a_0, a_2 - a_0, ..., a_i - a_0$ are linearly independent. First we need an auxiliary lemma.

Lemma 7.3.3 *The point a_{i+1} as defined in step 5 of Procedure 7.3.1 satisfies $a_{i+1}^T p_{i+1} \geq 0$.*

Proof:
x_{i+1} is an optimal argument for $L(\lambda, \mu)$. From this and the definitions of $\begin{pmatrix} \lambda \\ \mu \end{pmatrix}$ and a_{i+1} in step 5 it follows that

$$\begin{aligned} L(\lambda, \mu) &= f(u(x_{i+1}), c(x_{i+1})) - \lambda^T u(x_{i+1}) - \mu^T c(x_{i+1}) \\ &= f(u(x_{i+1}), c(x_{i+1})) - \hat{\lambda}^T u(x_{i+1}) - \hat{\mu}^T c(x_{i+1}) + \epsilon p_{i+1}^T a_{i+1}. \end{aligned} \tag{7.32}$$

If a component j of $\hat{\lambda}$ is zero then the corresponding component of p_{i+1} is less than or equal to zero because B (defined in step 2) contains a unit vector e_j with 1 as j-th component and p_{i+1} satisfies $e_j^T p_{i+1} \leq 0$ (required in step 4). Hence (λ, μ) is dual feasible. By the dual optimality of $(\hat{\lambda}, \hat{\mu})$

$$L(\lambda, \mu) \geq L(\hat{\lambda}, \hat{\mu})$$

Inserting (7.32) in LHS gives

$$\begin{aligned} & f(u(x_{i+1}), c(x_{i+1})) - \hat{\lambda}^T u(x_{i+1}) - \hat{\mu}^T c(x_{i+1}) + \epsilon p_{i+1}^T a_{i+1} \quad \geq \quad L(\hat{\lambda}, \hat{\mu}) \\ \Leftrightarrow \quad & \epsilon p_{i+1}^T a_{i+1} \geq \quad L(\hat{\lambda}, \hat{\mu}) \quad - \quad (f(u(x_{i+1}), c(x_{i+1})) - \hat{\lambda}^T u(x_{i+1}) - \hat{\mu}^T c(x_{i+1})) \end{aligned}$$

By definition of $L(\hat{\lambda}, \hat{\mu})$ we have RHS≥ 0.

$$\Rightarrow \qquad \epsilon p_{i+1}^T a_{i+1} \geq 0$$

Since $\epsilon > 0$ it implies $p_{i+1}^T a_{i+1} \geq 0$. $\square$.

Theorem 7.3.4 *The set of vectors $\{a_1 - a_0, a_2 - a_0, ..., a_i - a_0\}$, where a_j for $0 \leq j \leq i$ are defined in steps 1 and 5 of Procedure 7.3.1, is a set of linearly independent vectors.*

Proof:

The proof is done by induction. For the case $i = 1$ it must be shown that $a_1 - a_0 \neq 0$, *i.e.* that $a_1 \neq a_0$. This will be shown by contradiction. Assume

$a_1 = a_0$. We have $a_0^T p_1 < 0$ from the requirement in step 4. But from Lemma 7.3.3 $a_1^T p_1 \geq 0$. Contradiction.

Now assume that the theorem holds for some $i \geq 1$. It must be shown that if Procedure 7.3.1 does not terminate before defining a_{i+1} then $a_{i+1} - a_0$ is linearly independent from $a_1 - a_0, a_2 - a_0, ..., a_i - a_0$. Let's prove this, again, by contradiction.

$$\text{If} \qquad a_{i+1} - a_0 = \sum_{j=1}^{i} \alpha_j (a_j - a_0)$$

for some scalars α_j then

$$p_{i+1}^T (a_{i+1} - a_0) = \sum_{j=1}^{i} \alpha_j p_{i+1}^T (a_j - a_0). \tag{7.33}$$

LHS of (7.33) > 0 since $p_{i+1}^T a_{i+1} \geq 0$ by Lemma 7.3.3 and $p_{i+1}^T a_0 < 0$ by Lemma 7.3.2 and by step 4 of Procedure 7.3.1 which ensures that p_{i+1} satisfies the conditions of Lemma 7.3.2.
RHS of (7.33) $= 0$ since $p_{i+1}^T (a_j - a_0) = 0$ for $1 \leq j \leq i$ by the condition on p_{i+1} in step 4. (Remember that $(a_j - a_0) \in D_i$ for $1 \leq j \leq i$.) Contradiction. □.

Corollary 7.3.5 *The vectors a_0, a_1, ...,a_i are all different, where a_j for $0 \leq j \leq i$ are defined in steps 1 and 5 of Procedure 7.3.1.*

Proof:
By Theorem 7.3.4 $a_1 - a_0$, $a_2 - a_0$,..., $a_i - a_0$ are linearly independent. This implies that $a_j - a_0 \neq 0$ for $1 \leq j \leq i$. This means that a_0 is different from a_1, a_2,..., a_i. Now assume that $a_k = a_j$ for some j,k satisfying

$$1 \leq j \leq i,$$
$$1 \leq k \leq i,$$
$$j \neq k.$$

Then $a_k - a_0 = a_j - a_0$ and hence $a_1 - a_0$, $a_2 - a_0$,..., $a_i - a_0$ are linearly dependent. Contradiction to Theorem 7.3.4, hence $a_k \neq a_j$. □.

Corollary 7.3.5 is important because it shows that the perturbation method of Procedure 7.3.1 is efficient in the sense that every perturbation finds a new point of the $(\overline{u}, \overline{c}, \overline{f})$ space where (7.29) holds as equality. This implies that every perturbation finds a new vector $\hat{x} \in S$ where S is defined in (7.27)

7.4 Conclusions

The main result of this chapter is the perturbation procedure Procedure 7.3.1. The usual dual method for solving problem (7.1)-(7.4) solves the dual problem (7.6),(7.7) and then accepts $x(\hat{\lambda}, \hat{\mu})$ as the optimal point of problem (7.1)-(7.4), where $x(\hat{\lambda}, \hat{\mu})$ is a computed point satisfying (7.26) and $(\hat{\lambda}, \hat{\mu})$ is a solution point of the dual problem. If $x(\hat{\lambda}, \hat{\mu})$ is feasible and complementary slack then by Lagrange's Sufficiency Theorem $x(\hat{\lambda}, \hat{\mu})$ solves problem (7.1)-(7.4).

In the case that $x(\hat{\lambda}, \hat{\mu})$ is not feasible and complementary slack then typically there exist several points which are equally likely to be good near optimal points to problem (7.1)-(7.4). Procedure 7.3.1 discovers several of these near optimal points (at most it finds $m + k$ points). The usual dual method would compute only one of these near optimal points.

Chapter 8

A model for the Lagoven problem

8.1 Prototype hybrid algorithm

The Lagoven problem can be formulated as follows:

$$\begin{aligned} \text{maximize} \quad & f(x,v) && (8.1)\\ \text{subject to} \quad & u_1(x,v) \leq 0, && (8.2)\\ & u_2(x,v) \leq 0, && (8.3)\\ & c_1(x,v) = 0, && (8.4)\\ & c_2(x,v) = 0, && (8.5)\\ & x \in X, v \in V && (8.6) \end{aligned}$$

where $X \subset \mathbb{R}^{n_1}, V \subset \mathbb{R}^{n_2}$, the functions f, u_1, u_2, c_1, c_2 are $f : \mathbb{R}^{n_1+n_2} \to \mathbb{R}$, $u_1 : \mathbb{R}^{n_1+n_2} \to \mathbb{R}^{k_1}$, $u_2 : \mathbb{R}^{n_1+n_2} \to \mathbb{R}^{k_2}$, $c_1 : \mathbb{R}^{n_1+n_2} \to \mathbb{R}^{m_1}$, $c_2 : \mathbb{R}^{n_1+n_2} \to \mathbb{R}^{m_2}$. Define the following Lagrangian

$$\begin{aligned} L_v(\lambda,\mu) := \max_{x \in X} \{ & f(x,v) - \lambda^T u_1(x,v) - \mu^T c_1(x,v) \\ & : u_2(x,v) \leq 0, c_2(x,v) = 0 \} \end{aligned} \tag{8.7}$$

where $\lambda \in \mathbb{R}^{k_1}$, $\mu \in \mathbb{R}^{m_1}$. This means that only constraints (8.2) and (8.4) are relaxed. Suppose that there exists a method to evaluate $L_v(\lambda,\mu)$ for every λ, μ and $v \in V$. If $v \in V$ is fixed then it is possible to use the dual method to find a good x for problem (8.1)-(8.6). ie. pick $x \in S_v$ where

$$\begin{aligned} S_v := \{ x : x \in X, f(x,v) - \hat{\lambda}^T u_1(x,v) - \hat{\mu}^T c_1(x,v) = L_v(\hat{\lambda},\hat{\mu}), \\ u_2(x,v) \leq 0, \quad c_2(x,v) = 0 \} \end{aligned} \tag{8.8}$$

and where $(\hat{\lambda},\hat{\mu})$ solves the dual problem

$$\begin{aligned} \text{minimize} \quad & L_v(\lambda,\mu) && (8.9)\\ \text{subject to} \quad & \lambda \geq 0. \end{aligned}$$

Suppose problem (8.1)-(8.6) is nonlinear and continuous with a lot of local optima. An algorithm will be presented to find a good local optimum (x, v) using the dual method and a local optimization procedure (like SQP or SLP). The local optimization procedure used is one which minimizes the sum of constraint violations in the case that it can not find a feasible point. If the dual method is done with a numerical method and if the set S_v has more than one element then usually the numerical method will only return one element from S_v as the result x. So for the below algorithm let's assume that there is a numerical method to evaluate $L_v(\lambda, \mu)$, but this method always returns exactly one value for x which achieves the maximum in the RHS of (8.7). Next a basic algorithm is shown which combines a local search method applied to problem (8.1)-(8.6) with the dual method given by (8.7), (8.8), (8.9).

Algorithm 8.1.1

1) choose an integer $n > 0$.

2) Use some method to obtain starting values for the vectors x, v and store these as x_0, v_0 respectively.

3) For $i = 1$ to n do

4) With $v = v_{i-1}$ apply the dual method given by (8.9). The result is an optimal dual vectors $(\hat{\lambda}, \hat{\mu})$. Apply the perturbation procedure (Procedure 7.3.1) to $(\hat{\lambda}, \hat{\mu})$. The result is a set

$$E = \{\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_r\}$$

where $1 \leq r \leq k_1 + m_1$ and $E \subset S_{v_{i-1}}$.

5) For all j with $1 \leq j \leq r$ do a local optimization of problem (8.1)-(8.6) with $(\tilde{x}_j, v_{i-1})$ as a starting point. The result are locally optimal points $(\hat{x}_j, \hat{v}_j)$. (If the local optimization routine can not find a feasible point, it minimizes the sum of constraint violations instead.) From the points $(\hat{x}_1, \hat{v}_1), ..., (\hat{x}_r, \hat{v}_r)$ pick the best one with respect to constraint satisfaction and objective value and store it as (x_i, v_i).

6) If $v_i = v_j$ for some $j < i$ then go to step 8.

7) continue i.

8) From the points $(x_0, v_0), ..., (x_i, v_i)$ pick the best one with respect to constraint satisfaction and objective value and return it as the result to problem (8.1)-(8.6). Stop.

Steps 1 and 2 are initialisation steps. Essentially, within the $i-$loop (steps 3-7) dual optimization and local optimization are alternatingly applied. The idea is that the dual method in step 4 is often capable of 'escaping' from a bad local optimum. The algorithm produces a sequence of points $(x_0, v_0), (x_1, v_1), \ldots, (x_i, v_i)$ where $i \leq n$. Each of these points corresponds either to a local optimum of problem (8.1)-(8.6) or to a point which locally minimizes the sum of constraint violations in (8.2)-(8.6). At the end of the algorithm the best local optimum produced is accepted as the final result to problem (8.1)-(8.6). If the stopping condition in step 6 is fulfilled but the algorithm were to carry on then in the next iteration the algorithm would find $(x_{i+1}, v_{i+1}) = (x_{j+1}, v_{j+1})$ with $j < i$, and it would just keep on repeating previous computation. Usually, if the stopping condition in step 6 is fulfilled then $j = i - 1$ and (x_i, v_i) corresponds to a 'good' local optimum, where, of course, 'good' local optimum means a local optimum with a relatively high objective value.

8.2 Description of the Lagoven problem

In Lake Maracaibo in Venezuela there is a big oil production field, its description is the topic of this chapter. The operation of this oil production field gives rise to a non-convex, non-continuous optimization problem. A simplified model and ways to find good solutions to the corresponding optimization problem will be presented.

Figure 8.1 shows the simplified architecture of a oil production field. The lines in the figure stand for pipes, wells are represented by a circle. To each well gas is injected from one gas manifold. The injection of gas into a well has the effect of increasing the production rate of the well. The change of the production rate of the well depends on the injection rate of gas. Gas manifolds are represented by squares below the circles representing wells. A mixture of oil , water and gas comes out of a well and is delivered to a flowstation. Flowstations are represented by squares above the circles representing wells. At a flowstation water and oil is separated from the gas, which is going back into the system. Each flowstation has a limit on the rate of water that it can separate. Each flowstation i is operated at its own pressure $p_{f,i}$. (The index f stands for flowstation.) There are k flowstations. The rate of gas flow from flowstation i back into the system is $q_{f,i}$. (As before, the index f stands for flowstation.) The gas from the flowstations comes together at a node where the pressure is p_1. From there the gas flows to the compressor. The rate of gas flow to the compressor is q_1. At the entry point to the compressor the pressure is p_2. The compressor burns some of the incoming gas in

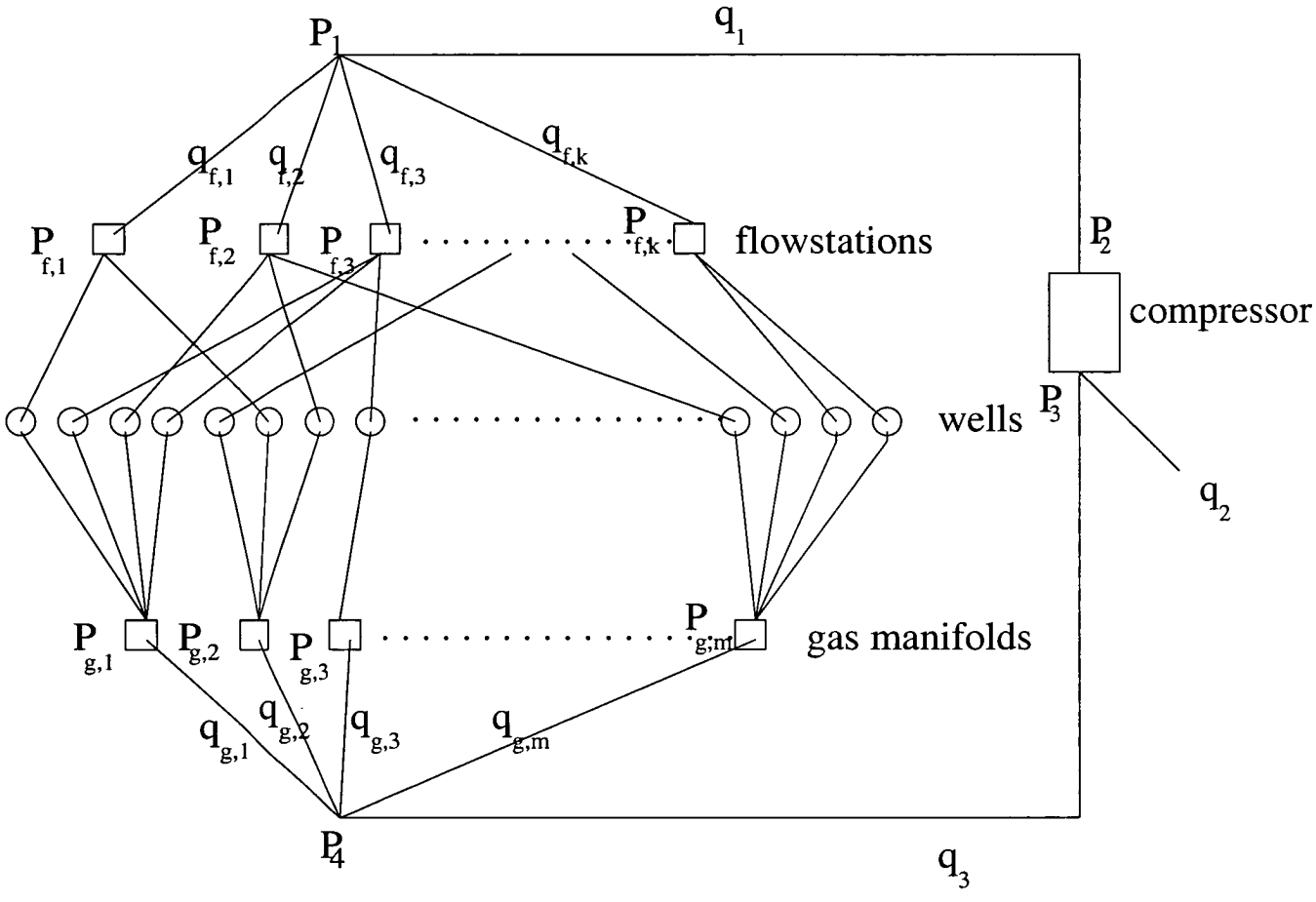

Figure 8.1: Model of the oil field

order to increase the pressure. Where the gas leaves the compressor, the pressure is p_3. Some of the out-flowing gas leaves the system and is sold, this flow of gas is q_2. The price for one unit of gas is c. The out-flowing gas staying in the system is q_3. This gas flows to a node where the pressure is p_4. From this node the gas is delivered to m different gas manifolds. The rate of flow of gas to gas manifold i is $q_{g,i}$. (The index g stands for gas manifold.) Gas manifold i is operated at pressure $p_{g,i}$. There are n wells and they are numbered from left to right. Wells $1, 2...n_1$ receive the lift gas from gas manifold 1. Wells $n_1 + 1, ..., n_1 + n_2$ receive the lift gas from gas manifold 2. If

$$z_1 := 0, \quad z_i := \sum_{j=1}^{i-1} n_j \quad \text{for } i \geq 2,$$

then wells $z_i + 1, ..., z_i + n_i$ receive lift gas from gas manifold i. (Notice that $\sum_{j=1}^{m} n_j = n$) The rate of lift gas injection into well i is x_i. $(x_i \geq 0)$.

Well i is connected to flowstation $\pi(i)$. ie π is a function from $\{1, 2, ...n\}$ to $\{1, 2, ..., k\}$. Well i is connected to gas manifold $\phi(i)$. ie ϕ is a function from $\{1, 2, ...n\}$ to $\{1, 2, ..., m\}$. (From what was said about the connection of gas manifolds with wells, it follows that $\phi(i) = j$ for $z_j < i \leq z_j + n_j$.) The flow rate of oil coming out of well i is

$$f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}).$$

The flow rate of gas coming out of well i is

$$g_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}).$$

The flow rate of water coming out of well i is

$$s_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}).$$

ie. f_i, g_i and s_i are functions $\mathbb{R}^3 \to \mathbb{R}$ for $1 \leq i \leq n$. Also, the functions f_i, g_i and s_i are always non-negative. Along the pipes pressure drop takes place. The pressure drop in the pipe connecting flowstation i with the node where the pressure is p_1 can be described by

$$w_{f,i}(p_{f,i}, q_{f,i}, p_1) = 0$$

$w_{f,i}$ is a function relating the pressures at the two ends of the pipe with the flow of gas through it. It is a continuous function $\mathbb{R}^3 \to \mathbb{R}$. (The index f, again, stands just for flowstation.) Similarly, the pressure drop in the pipe connecting the node where the pressure is p_1 with the compressor can be described by

$$w_1(p_1, q_1, p_2) = 0,$$

w_1 is a continuous function $\mathbb{R}^3 \to \mathbb{R}$. The pressure drop in the pipe connecting the compressor with the node where the pressure is p_4 can be described by

$$w_2(p_3, q_3, p_4) = 0,$$

w_2 is a continuous function $\mathbb{R}^3 \to \mathbb{R}$. The pressure drop in the pipe connecting the node where the pressure is p_4 with gas manifold i can be described by

$$w_{g,i}(p_4, q_{g,i}, p_{g,i}) = 0$$

(The index g stands for gas manifold.) $w_{g,i}$ is a continuous function $\mathbb{R}^3 \to \mathbb{R}$. There is, of course, also pressure drop in the pipes connecting wells to flowstations and to gas manifolds, but these pressure drops are already incorporated in the functions f_i, g_i and s_i for $i = 1, 2, ..., n$, hence there is no need to include these pressure drops explicitly in the mathematical formulation. The compressor burns some of the incoming gas in order to increase the pressure. The action of the compressor can be described by

$$h(p_2, p_3, q_1, q_4) = 0$$

where h is a continuous function $\mathbb{R}^4 \to \mathbb{R}$ and q_4 is the rate of gas being burnt by the compressor. From the mass balance at the compressor it follows that $q_4 = q_1 - q_2 - q_3$. Before the whole optimization problem is formulated, one more object must be defined. Let

$$Q(j) = \{i : 1 \leq i \leq n, \pi(i) = j\}$$

So $Q(j)$ is the set of indices of wells which are connected to flowstation j. The optimization problem will be formulated now:

$$\text{maximize} \quad cq_2 + \sum_{i=1}^{n} f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \tag{8.10}$$

$$\text{subject to} \quad \sum_{i \in Q(j)} s_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - S_j \leq 0 \quad \forall j : 1 \leq j \leq k, \tag{8.11}$$

$$\sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - q_{f,j} = 0 \quad \forall j : 1 \leq j \leq k, \tag{8.12}$$

$$w_{f,i}(p_{f,i}, q_{f,i}, p_1) = 0 \quad \forall i : 1 \leq i \leq k, \tag{8.13}$$

$$\sum_{i=1}^{k} q_{f,i} \quad - q_1 = 0, \tag{8.14}$$

$$w_1(p_1, q_1, p_2) = 0, \tag{8.15}$$

$$q_1 - q_2 - q_3 - q_4 = 0, \tag{8.16}$$

$$h(p_2, p_3, q_1, q_4) = 0, \tag{8.17}$$

$$w_2(p_3, q_3, p_4) = 0, \tag{8.18}$$

$$\sum_{i=1}^{m} q_{g,i} \quad - q_3 = 0, \tag{8.19}$$

$$w_{g,i}(p_4, q_{g,i}, p_{g,i}) = 0 \quad \forall i : 1 \leq i \leq m, \tag{8.20}$$

$$\sum_{i=z_j+1}^{z_j+n_j} x_i \quad - q_{g,j} = 0 \quad \forall j : 1 \leq j \leq m, \tag{8.21}$$

$$q_1, q_2, q_3, q_4 \geq 0, \quad x_i, q_{f,i}, q_{g,i} \geq 0 \quad \forall i \tag{8.22}$$

(8.10) is the objective function, (8.11) describes the separation constraint at the flowstations where S_j is the limit on the rate of water arriving at flowstation j, (8.12) describes the conservation of gas at the flowstations, (8.13) describes the pressure drop in the pipes connecting a flowstation with the point where the pressure is p_1, (8.14) describes the conservation of gas at the point where the pressure is p_1, (8.15) describes the pressure drop in the pipe connecting the point where the pressure is p_1 with the point where the pressure is p_2, (8.16) describes the conservation of gas at the compressor, (8.17) describes the constraint describing the influence of the compressor (ie. the burning of some gas in exchange for an increase in pressure), (8.18) describes the pressure drop constraint of the pipe connecting the points where the pressures are p_3 and p_4 respectively, (8.19) describes the conservation of gas at the point where the pressure is p_4, (8.20) describes the pressure drop in the pipes connecting the point where the pressure is p_4 with gas manifolds, and finally, (8.21) describes the conservation of gas at the gas manifolds. This completes the circle of the network. (8.22) describes the positivity constraints for the flow variables.

The functions f_i, s_i and g_i are typically non-continuous. For fixed $a, b \in \mathbb{R}$, the functions $f_i(x, a, b), s_i(x, a, b)$ and $g_i(x, a, b)$, which are now functions in x only, have one common point where the functions are discontinuous.

$$\text{Let this point be } x_i^*(a, b). \tag{8.23}$$

x_i^* is continuous in a and b. When a, b are fixed the functions $f_i(x, a, b)$,$s_i(x, a, b)$, $g_i(x, a, b)$ are continuous in x on the intervals $(-\infty, x_i^*)$ and $[x_i^*, \infty)$. The other functions in the problem are everywhere continuous. The variables in this problem are $x_1, x_2, ..., x_n, p_{g,1}, p_{g,2}, \cdots, p_{g,m}, q_{g,1}, q_{g,2}, \cdots, q_{g,m}, p_{f,1}, p_{f,2}, \cdots, p_{f,k}, q_{f,1}, q_{f,2}, \cdots, q_{f,k}$, $p_1, p_2, p_3, p_4, q_1, q_2, q_3, q_4$. Let these variables be grouped together in vectors as x, p_g, q_g, p_f, q_f, p, q. The obvious way to tackle this problem is to try a local optimization technique like sequential linear programming or sequential quadratic programming. However, the discontinuities of the functions f_i, s_i, g_i make this approach problematic because one has to decide at which side of the discontinuity the first argument of these functions shall be. It is easy to see that a lot of

combinations arise. A hybrid algorithm including a local optimization technique (ie. sequential linear or quadratic programming), dynamic programming and the dual method will be proposed. This hybrid algorithm is of the form described in section 8.1.

8.3 Description and solution approach of a subproblem

In this section a subproblem of the optimization problem of the previous section will be described and a dual algorithm for its solution will be discussed. Consider the following subproblem:

$$\text{maximize}_{\{x_i\}_{i=1}^n} \quad \sum_{i=1}^{n} f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \tag{8.24}$$

$$\text{subject to} \sum_{i \in Q(j)} s_i(x_i, p_{g,\phi(i)}, p_{f,j}) - S_j \leq 0 \quad \forall j : 1 \leq j \leq k, \tag{8.25}$$

$$\sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) - q_{f,j} = 0 \quad \forall j : 1 \leq j \leq k, \tag{8.26}$$

$$\sum_{i=z_j+1}^{z_j+n_j} x_i = q_{g,j} \quad \forall j : 1 \leq j \leq m, \tag{8.27}$$

$$x_i \geq 0 \quad \forall i \tag{8.28}$$

The optimization in this subproblem is in the variables $\{x_i\}_{i=1}^n$ only, all other variables are fixed. In terms of Figure 8.1, this subproblem just looks at the area between the gas manifolds and the flowstations. Notice that constraint (8.25) is equivalent to (8.11) and (8.26) is equivalent to (8.12). (8.27) is identical to (8.21). For the dual algorithm a Lagrangian function will be defined where constraints (8.25) and (8.26) will be relaxed.

$$\begin{aligned} L(\lambda, \mu) := \max_{\{x_i\}_{i=1}^n} \Bigg\{ & \sum_{i=1}^{n} f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \\ & - \sum_{j=1}^{k} \lambda_j \Big(\sum_{i \in Q(j)} s_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - S_j \Big) \\ & - \sum_{j=1}^{k} \mu_j \Big(\sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - q_{f,j} \Big) \quad : \\ & \sum_{i=z_j+1}^{z_j+n_j} x_i = q_{g,j} \quad \forall j : 1 \leq j \leq m, \text{ and } x_i \geq 0 \quad \forall i \Bigg\} \end{aligned} \tag{8.29}$$

λ and μ are k dimensional vectors, their components are given by $\lambda^T = (\lambda_1, \lambda_2, ..., \lambda_k)$ and $\mu^T = (\mu_1, \mu_2, ..., \mu_k)$. (and all $\lambda_i, \mu_i \in I\!R$)

For every $i : 1 \leq i \leq n$ there exists exactly one $j : 1 \leq j \leq k$ such that $i \in Q(j)$. From the definition of $Q(j)$ it follows that

$$i \in Q(j) \Leftrightarrow \pi(i) = j.$$

Hence the following equivalence holds:

$$\begin{aligned}
&\sum_{j=1}^{k} \lambda_j \Big(\sum_{i \in Q(j)} s_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - S_j \Big) \\
= &\sum_{j=1}^{k} \sum_{i \in Q(j)} \lambda_j s_i(x_i, p_{g,\phi(i)}, p_{f,j}) - \sum_{j=1}^{k} \lambda_j S_j \\
= &\sum_{j=1}^{k} \sum_{i \in Q(j)} \lambda_{\pi(i)} s_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) - \sum_{j=1}^{k} \lambda_j S_j \\
= &\sum_{i=1}^{n} \lambda_{\pi(i)} s_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) - \sum_{j=1}^{k} \lambda_j S_j \qquad (8.30)
\end{aligned}$$

Similarly:

$$\begin{aligned}
&\sum_{j=1}^{k} \mu_j \Big(\sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - q_{f,j} \Big) \qquad (8.31) \\
= &\sum_{i=1}^{n} \mu_{\pi(i)} g_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) - \sum_{j=1}^{k} \mu_j q_{f,j}
\end{aligned}$$

Inserting (8.30) and (8.31) in (8.29):

$$\begin{aligned}
L(\lambda, \mu) = \max_{\{x_i\}_{i=1}^{n}} \Bigg\{ &\sum_{i=1}^{n} f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \\
&- \sum_{i=1}^{n} \lambda_{\pi(i)} s_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) + \sum_{j=1}^{k} \lambda_j S_j \\
&- \sum_{i=1}^{n} \mu_{\pi(i)} g_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) + \sum_{j=1}^{k} \mu_j q_{f,j} \quad : \\
\sum_{i=z_j+1}^{z_j+n_j} x_i = q_{g,j} \quad &\forall j : 1 \leq j \leq m, \text{ and } x_i \geq 0 \quad \forall i \Bigg\} \qquad (8.32)
\end{aligned}$$

$L(\lambda, \mu)$ can be evaluated numerically by dynamic programming. In order to see how let's define

$$T_j(\lambda, \mu) := \max_{\{x_i\}_{i=z_j+1}^{z_j+n_j}} \Bigg\{ \sum_{i=z_j+1}^{z_j+n_j} f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)})$$

$$- \sum_{i=z_j+1}^{z_j+n_j} \lambda_{\pi(i)} s_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)})$$

$$- \sum_{i=z_j+1}^{z_j+n_j} \mu_{\pi(i)} g_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \quad :$$

$$\left. \sum_{i=z_j+1}^{z_j+n_j} x_i = q_{g,j}, \text{ and } x_i \geq 0 \quad \forall i : z_j + 1 \leq i \leq z_j + n_j \right\} \tag{8.33}$$

$T_j(\lambda, \mu)$ is well defined for $j : \quad 1 \leq j \leq m$.
Now define

$$\hat{f}_i(x_i) := f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) - \lambda_{\pi(i)} s_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) - \mu_{\pi(i)} g_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \tag{8.34}$$

Then

$$T_j(\lambda, \mu) = \max_{\{x_i\}_{i=z_j+1}^{z_j+n_j}} \left\{ \sum_{i=z_j+1}^{z_j+n_j} \hat{f}_i(x_i) \quad : \right.$$

$$\left. \sum_{i=z_j+1}^{z_j+n_j} x_i = q_{g,j}, \text{ and } x_i \geq 0 \quad \forall i : z_j + 1 \leq i \leq z_j + n_j \right\} \tag{8.35}$$

From this it can be seen that it is possible to enumerate $T_j(\lambda, \mu)$ approximately by dynamic programming using n_j stages and one-dimensional continuous action and state spaces. The finer the discretisation one uses the better the numerical solution will be. Further, since the functions $\hat{f}_i$ are continuous and differentiable everywhere except at one point the Fritz-John action elimination can be applied which speeds up the algorithm. From the previous definitions it can be seen that

$$L(\lambda, \mu) = \sum_{j=1}^{m} T_j(\lambda, \mu) + \sum_{j=1}^{k} (\lambda_j S_j + \mu_j q_{f,j}) \tag{8.36}$$

Hence it is clear that $L(\lambda, \mu)$ can be evaluated numerically using dynamic programming. $L(\lambda, \mu)$ is a convex function, this has been proved in Lemma 7.1.1. It is important to recognise that the evaluation of $L(\lambda, \mu)$ can not be done analytically, it must be done numerically (using dynamic programming). Even though $L(\lambda, \mu)$ is convex, in general $L(\lambda, \mu)$ is non-smooth. A subtangent of $L(\lambda, \mu)$ is available. Next a subtangent of $L(\lambda, \mu)$ and its derivation will be given.

Theorem 8.3.1 *Let* $(\hat{\lambda}, \hat{\mu}) \in \mathbb{R}^k \times \mathbb{R}^k$, *let* $\hat{x}$ *satisfy (8.39) and* p *be a* $2k$*-dimensional vector with components*

$$p_j = S_j - \sum_{i \in Q(j)} s_i(\hat{x}_i, p_{g,\phi(i)}, p_{f,j}),$$

$$p_{k+j} = q_{f,j} - \sum_{i \in Q(j)} g_i(\hat{x}_i, p_{g,\phi(i)}, p_{f,j})$$

for $1 \leq j \leq k$. *Then* p *is a subtangent vector of the subdifferential set* $\partial L(\hat{\lambda}, \hat{\mu})$, *i.e.*

$$L(\lambda, \mu) \geq L(\hat{\lambda}, \hat{\mu}) + p^T \begin{pmatrix} \lambda - \hat{\lambda} \\ \mu - \hat{\mu} \end{pmatrix}$$

for all $\lambda \in \mathbb{R}^k, \mu \in \mathbb{R}^k$.

Proof:

The theorem is a direct consequence of Theorem 7.1.6. In order to apply results of section 7.1 it must be shown that problem (8.24)-(8.28) is of the problem form (7.1)-(7.4) and that the definition of $L(\lambda, \mu)$ in (8.29) is of the form (7.5):

(8.24) is of the form (7.1), (8.25) is of the form (7.2) and (8.26) is of the form (7.3) (setting for $m = k$) since in (8.24)-(8.26) only $\{x_i\}_{i=1}^n$ are variables and p_g, p_f, q_f and $\{S_j\}_{j=1}^k$ are fixed parameters. (8.27) and (8.28) are of the form (7.4) since in the definition of $L(\lambda, \mu)$ in (8.29) the constraints (8.27) and (8.28) are not relaxed. (8.27) and (8.28) can be thought of as conditions defining a set X. Having established correspondences between (8.24)-(8.28) and (7.1)-(7.4) it is clear that the definition of $L(\lambda, \mu)$ in (8.29) is of the form (7.5) and hence that all results, methods and procedures of section 7.1 can be applied to the subproblem (8.24)-(8.28).

In the previous paragraph it has been said that (8.25) is of the form (7.2). More explicitly, this means that $u(x)$ in (7.2) is the vector with components

$$u_j(x) = \sum_{i \in Q(j)} s_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - S_j \qquad \text{for } 1 \leq j \leq k.$$

Also (8.26) being of the form (7.3) with $m = k$ in (7.3) means that $c(x)$ in (7.3) is the vector with components

$$c_j(x) = \sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - q_{f,j} \qquad \text{for } 1 \leq j \leq k.$$

Using these expressions for $u(x)$ and $c(x)$ in Theorem 7.1.6 yields Theorem 8.3.1. □.

$L(\lambda, \mu)$ is well defined. The dual problem of subproblem (8.24)-(8.28) is

$$\begin{aligned} &\text{minimize} && L(\lambda, \mu) && (8.37) \\ &\text{subject to} && \lambda \geq 0. && (8.38) \end{aligned}$$

This problem can be solved using a local optimization technique. A cutting plane method is suitable for solving the dual problem because $L(\lambda, \mu)$ is convex and

because at every point (λ, μ) the evaluation of $L(\lambda, \mu)$ also provides an optimal argument $x(\lambda, \mu)$ in the RHS of (8.32) and this in turn provides a cutting plane (Theorem 8.3.1). Since $L(\lambda, \mu)$ is a convex function every local optimum is a global optimum.

Let $\hat{\lambda}$ and $\hat{\mu}$ be the solution vectors to problem (8.37) and (8.38), their components be given by $\hat{\lambda}^T = (\hat{\lambda}_1, \hat{\lambda}_2, ..., \hat{\lambda}_k)$ and $\hat{\mu}^T = (\hat{\mu}_1, \hat{\mu}_2, ..., \hat{\mu}_k)$. Let $\hat{x}$ be an argument in (8.29) which achieves the maximum when λ, μ are replaced by $\hat{\lambda}, \hat{\mu}$. The components of $\hat{x}$ are given by $\hat{x}^T = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_n)$. From the last two sentences:

$$\hat{x} \in \arg \max_{\{x_i\}_{i=1}^n} \left\{ \sum_{i=1}^{n} f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \right.$$
$$- \sum_{j=1}^{k} \hat{\lambda}_j \Big(\sum_{i \in Q(j)} s_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - S_j \Big)$$
$$- \sum_{j=1}^{k} \hat{\mu}_j \Big(\sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - q_{f,j} \Big) \quad :$$
$$\left. \sum_{i=z_j+1}^{z_j+n_j} x_i = q_{g,j} \quad \forall j : 1 \leq j \leq m, \text{ and } x_i \geq 0 \quad \forall i \right\} \tag{8.39}$$

If $x = \hat{x}$ satisfies (8.25) and (8.26), and also satisfies the complementary slackness condition

$$\hat{\lambda}_j \Big(\sum_{i \in Q(j)} s_i(x_i, p_{g,\phi(i)}, p_{f,j}) - S_j \Big) = 0 \quad \forall j : 1 \leq j \leq k$$

then $\hat{x}$ solves subproblem (8.24)-(8.28) by Lagrange's sufficiency theorem. However, in general there is no guarantee that the conditions of the last sentence are fulfilled by $\hat{x}$. $\hat{x}$ is an element of a set of optimal arguments which often has more than one element. In order to find an element $\hat{x}$ from the set in (8.39) which violates constraints (8.25), (8.26) only a little the perturbation method of Procedure 7.3.1 can be applied. In order to see how this can be done exactly it is useful to have in mind how subproblem (8.24)-(8.28) can be put into the problem form (7.1)-(7.4) to which Procedure 7.3.1 applies the perturbation method. In the proof of Theorem 8.3.1 it is described how (8.24)-(8.28) corresponds to (7.1)-(7.4).

In section 7.1 it was always assumed that there exists a method for the evaluation of $L(\lambda, \mu)$ for every choice of (λ, μ) which also returns an optimal argument $x(\lambda, \mu)$ satisfying (7.26). In this section it has been shown that DP provides such an evaluation method for $L(\lambda, \mu)$ as defined in (8.29).

8.4 Proposition of a hybrid algorithm

In this section a hybrid algorithm is presented for the Lagoven problem given by (8.10)- (8.22). It is based on Algorithm 8.1.1 in section 8.1. The vector v of section 8.1 corresponds to p_g, q_g, p_f, q_f, p, q. The various steps will be discussed in more detail after presenting the algorithm.

Algorithm 8.4.1

1) choose an integer $N > 0$.

2) Use some method to obtain starting values for $x, p_g, q_g, p_f, q_f, p, q$. Store these starting values as vector y_0 by setting

$$y_0 = \begin{pmatrix} x \\ p_g \\ q_g \\ p_f \\ q_f \\ p \\ q \end{pmatrix}.$$

3) For $k = 1$ to N do

4) With values of p_g, q_g, p_f, q_f taken from y_{k-1} solve the dual problem (8.37),(8.38) of subproblem (8.24)-(8.28). The result is an optimal dual vector $(\hat{\lambda}, \hat{\mu})$. Having solved the dual problem apply the perturbation method of Procedure 7.3.1 to $(\hat{\lambda}, \hat{\mu})$. The result is a finite ordered set E with r elements where every $\hat{x} \in E$ satisfies (8.39).

5) For each $\hat{x} \in E$ do a local optimization as follows:

   a) Use as starting point a vector y with x-component $\hat{x}$ and p_g, q_g, p_f, q_f, p, q components equal to those of y_{k-1}.

   b) Let I be the set of indices

$$I = \{i : \hat{x}_i \geq x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)})\}$$

   where the meaning of $x^*(a, b)$ is taken from (8.23).

   c) Apply a local optimization routine (SLP or SQP) to problem (8.10)-(8.22) with the additional constraints

$$x_i \geq x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)}) \quad \forall i \in I$$
$$x_i \leq x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)}) - \epsilon \quad \forall i \notin I$$

where $\epsilon > 0$ is a small tolerance parameter. If the algorithm can not find a feasible point then it minimizes the sum of constraint violations instead.

The result of these r local optimizations are locally optimal points $\tilde{y}_1, \tilde{y}_2, ..., \tilde{y}_r$. From these points pick the best one with respect to constraint satisfaction and objective value and store it as y_k.

6) If $y_k = y_j$ for some $j < k$ then go to step 8.

7) continue k.

8) From the points $y_0, ..., y_k$ pick the best one with respect to constraint satisfaction and objective value and return it as the result to problem (8.10)-(8.22). Stop.

Steps 1 and 2 are initialisation steps. Step 4 does an optimization of a subproblem with variables x only, the optimization method applied can cope with discontinuities of the functions f_i, g_i and s_i. Step 5 applies a local optimization procedure like SQP or SLP to the whole Lagoven problem and since local optimization algorithms cope badly with discontinuous points, additional constraints are being imposed to insure that the discontinuous points do not obstruct the local optimization process. The algorithm generates vectors $y_0, y_1, ..., y_k$ where $k \leq N$. Not all of these vectors necessarily correspond to feasible points of the Lagoven problem (8.10)-(8.22), but in practice usually they do. Those vectors which correspond to feasible points are locally optimal. Steps 4 and 5 amend each other in the following sense: Step 4 involves the dual method in combination with dynamic programming which is described in section 8.3 and step 5 involves a local optimization algorithm. The method of section 8.3 copes well with the discontinuities of the functions f_i, g_i, s_i, whereas local optimization algorithms cope badly with discontinuous points. Secondly, the method of section 8.3 only optimizes over the vector x and it is not good at satisfying all constraints exactly, whereas a local optimization algorithm optimizes over all variables (*i.e.* x, p_g, q_g, p_f, q_f, p, q) and generally it is good at satisfying all constraints exactly, given that the starting point is almost feasible.

This algorithm is very similar to Algorithm 8.1.1 in section 8.1. The dual method in the algorithm of section 8.1 is not specified whereas it is specified in this algorithm.

Recall that E is a set of optimal arguments $\hat{x}$ satisfying (8.39) where $(\hat{\lambda}, \hat{\mu})$ is a solution to the dual problem (8.37),(8.38) which is the dual problem to subproblem (8.24)-(8.28). E is an ordered set, the order being according to the sum

of constraint violations in (8.25),(8.26) by an element $\hat{x} \in E$. The first element $\hat{x} \in E$ is the one with the smallest sum of constraint violations, the last element $\hat{x}$ the one with the largest sum of constraint violations. It can be expected that an element $\hat{x} \in E$ with small sum of constraint violations is better as a starting point for the local optimization in step 5 than another element $x \in E$ with larger sum of constraint violations. This is a heuristic argument which Algorithm 8.4.1 does not rely on.

In Algorithm 8.4.1 the vectors p_g, q_g, p_f, q_f, p, q play the role of v in Algorithm 8.1.1.

8.4.1 Initial values for $x, p_g, q_g, p_f, q_f, p, q$

In this section a method to find initial values for x,p_g,q_g,p_f,q_f,p, q in step 2 of Algorithm 8.4.1 is presented. Basically the idea is to apply a local search method to a continuous problem which approximates the non-continuous problem (8.10)-(8.22).

In problem (8.10)-(8.22) in praxis the functions f_i, s_i, g_i, $w_{f,i}$, w_1, h, w_2, $w_{g,i}$ are functions with the characteristic that when all arguments are zero then the function value is zero. This implies that the point

$$\begin{pmatrix} x \\ p_g \\ q_g \\ p_f \\ q_f \\ p \\ q \end{pmatrix} = 0$$

is feasible for problem (8.10)-(8.22).

Remember that the functions f_i, s_i, g_i have discontinuities which is the reason why Algorithm 8.4.1 has been proposed instead of simply applying a local optimization routine to problem (8.10)-(8.22). In order to find initial values for x,p_g,q_g,p_f,q_f,p, q , which is needed in step 2 of Algorithm 8.4.1, do the following process:

1) Construct CONTINUOUS functions $\tilde{f}_i$, $\tilde{s}_i$, $\tilde{g}_i$ which approximate the functions f_i, s_i, g_i respectively. When all arguments are zero then $\tilde{f}_i$, $\tilde{s}_i$, $\tilde{g}_i$ shall take the value zero (like f_i, s_i, g_i do).

2) Apply a local optimization routine to problem (8.10)-(8.22) with the functions f_i, s_i, g_i being replaced by $\tilde{f}_i$, $\tilde{s}_i$, $\tilde{g}_i$ respectively. Use as starting

point

$$\begin{pmatrix} x \\ p_g \\ q_g \\ p_f \\ q_f \\ p \\ q \end{pmatrix} = 0.$$

3) The result of the previous step is a point which is locally optimal for problem (8.10)-(8.22) with the functions f_i, s_i, g_i being replaced by $\tilde{f}_i$, $\tilde{s}_i$, $\tilde{g}_i$ respectively. Let this point define the initial values of x, p_g, q_g, p_f, q_f, p, q in step 2 of Algorithm 8.4.1.

8.5 Perturbation and discontinuities

In this section a special situation is described. This kind of situation appears when solving the Lagoven problem (8.10)-(8.22) with Algorithm 8.4.1. In section 8.6 problems of Algorithm 8.4.1 will be discussed and removed. For the clearer understanding of section 8.6 it is necessary to be aware about the special case described in this section.

Consider the following optimization problem in two variables x and y.

$$\begin{aligned} \text{maximize} \quad & f(x,y) \\ \text{subject to} \quad & v(x) - y = 0, \\ x \geq 0 \quad , \quad & y \geq 0 \end{aligned} \tag{8.40}$$

where $x, y \in \mathbb{R}$, $f : \mathbb{R}^2 \to \mathbb{R}$, $v : \mathbb{R} \to \mathbb{R}$. Suppose that

$$\begin{aligned} v(0) &= 0, \\ v(x) &\geq a \qquad \text{for all } x > 0 \end{aligned}$$

where a is a nonzero positive real number. *i.e.* v is discontinuous at $x = 0$. Let this problem be solved by an algorithm which alternatingly solves (8.40) in x only using the dual method (with perturbation of the optimal dual variables) and then applies a local optimization in x and y, taking the solution of the previous dual method (local optimization) as the starting point of the local optimization (dual method). Suppose that at the beginning of a dual method $y = 0$. The dual method minimizes $L(\mu)$ where

$$L(\mu) := \max_{x \geq 0} \{f(x,y) - \mu(v(x) - y)\}$$

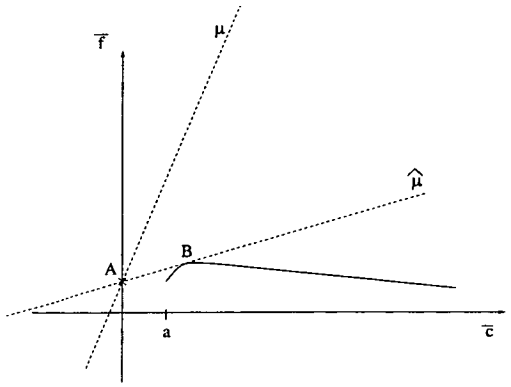

Figure 8.2:

Figure 8.2 shows what the $\overline{f}(\bar{c})$ plot could look like where

$$\overline{f}(\bar{c}) := \max_{x \geq 0}\{f(x,y) \quad : v(x) - y = \bar{c}\}$$

and $\overline{f}(\bar{c}) := -\infty$ if there exists no $x \geq 0$ such that $v(x) - y = \bar{c}$. Notice that in Figure 8.2 $\overline{f}(\bar{c}) = -\infty$ for $\bar{c} < 0$ and for $0 < \bar{c} < a$ because $v(x) - y = 0$ only for $x = 0$ and for no $x \geq 0$ is $0 < v(x) - y < a$. This is because of the assumptions made about $v(x)$ and because $y = 0$. $L(\mu)$ is minimized for all $\mu \geq \hat{\mu}$ as can be seen from Figure 8.2. Point A arises from $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$. It is desirable that the dual method not only discovers A but also B in order to get over the discontinuity of $v(x)$ at $x = 0$. If the minimization routine used for the minimization of $L(\mu)$ returns $\hat{\mu}$ as optimal solution then the perturbation method, which is the next step of the dual method, will discover B. If the minimization routine returns a $\mu >> \hat{\mu}$ as optimal solution then the perturbation method will not discover B. A good trick to ensure that the dual method discovers B is the following: replace y in the definition of $L(\mu)$ by $\hat{y}$ which satisfies:

$$\begin{aligned} &\text{if } y > 0 \quad \text{then} \quad \hat{y} = y, \\ &\text{if } y = 0 \quad \text{then} \quad 0 < \hat{y} < a. \end{aligned}$$

When $L(\mu)$ with this definition , *i.e.*

$$L(\mu) := \max_{x \geq 0}\{f(x,\hat{y}) - \mu(v(x) - \hat{y})\},$$

is minimized then the only optimal solution is $\hat{\mu}$ which can be seen from Figure 8.3. Notice that $\overline{f}(\bar{c})$ in Figure 8.3 is $\overline{f}(\bar{c})$ from Figure 8.2 shifted by $\hat{y}$ to the left. Since now the minimization routine for the minimization of $L(\mu)$ can only return $\hat{\mu}$ as optimal solution, the perturbation method will discover B when perturbing $\hat{\mu}$.

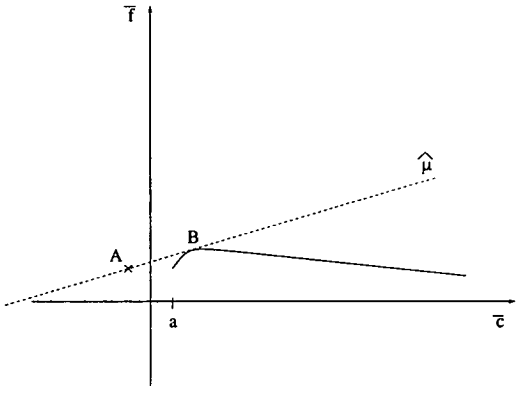

Figure 8.3:

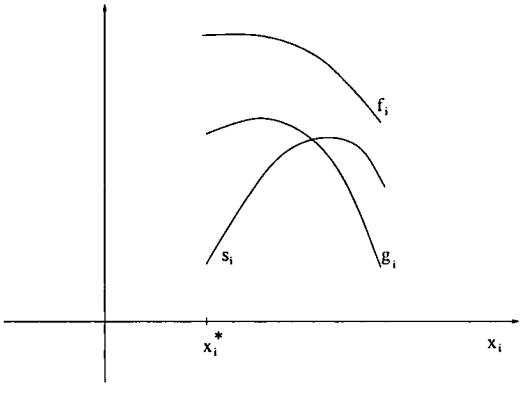

Figure 8.4:

8.6 Problems of the hybrid algorithm and how to remove them

Algorithm 8.4.1 of the last section is an algorithm for the solution of problem (8.10)-(8.22). It combines a local search method like SLP or SQP with the dual solution procedure of subproblem (8.24)-(8.28) which is described in section 8.3. In practice, when combining the dual solution procedure with the local search method some problems arise. All these problems have to do with the point $x_i^*(a,b)$ at which the functions $f_i(x,a,b)$, $s_i(x,a,b)$, and $g_i(x,a,b)$, when considered as functions of x only, are non-continuous. Figures 8.4-8.5 show some typical plots of these functions. The main characteristic to be noticed about the functions f_i, s_i, g_i is the following: when $x_i^*(a,b) > 0$ then for all $x < x_i^*(a,b)$

$$f_i(x,a,b) = 0, \tag{8.41}$$

$$s_i(x,a,b) = 0, \tag{8.42}$$

$$g_i(x,a,b) = 0. \tag{8.43}$$

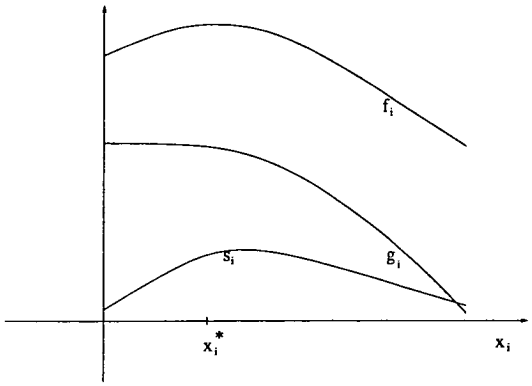

Figure 8.5:

Suppose that after a local optimization in Algorithm 8.4.1 $q_{g,j} = 0$ for some j and

$$x_i^*(a, b) > 0 \tag{8.44}$$

for all a, b and for all i with $\phi(i) = j$ (or equivalently $z_j + 1 \leq i \leq z_j + n_j$). *i.e.* for all wells i, which receive lift gas from gas manifold j, the discontinuity point x_i^* of the functions f_i, s_i, g_i is greater than zero for all choices of pressures at the gas manifold and at the flowstation which well i is connected to. Further suppose that at this stage Algorithm 8.4.1 is at a feasible point. Then constraints (8.21) and (8.22) imply that $x_i = 0$ for all i with $z_j + 1 \leq i \leq z_j + n_j$ (or equivalently $\phi(i) = j$). The next step in Algorithm 8.4.1 is to solve subproblem (8.24)-(8.28) with the dual method using DP for the evaluation of $L(\lambda, \mu)$. Since in the definition of $L(\lambda, \mu)$ the constraint (8.21) (the same as constraint (8.27)) appears as a non-relaxed constraint, DP actually only considers the zero allocation $x_i = 0$ for all wells i with $z_j + 1 \leq i \leq z_j + n_j$. This implies that the result of the dual method applied to subproblem (8.24)-(8.28) is a distribution x with $x_i = 0$ for $z_j + 1 \leq i \leq z_j + n_j$. The next step in Algorithm 8.4.1 is a local optimization applied to problem (8.10)-(8.22) with a starting point that has $x_i = 0$ for $z_j + 1 \leq i \leq z_j + n_j$. The local optimization is not able to cross the discontinuity point x_i^*. Hence the computed optimal point of the local optimization will satisfy

$$x_i < x_i^*(p_{g,j}, p_{f,\pi(i)}) \qquad \text{for } z_j + 1 \leq i \leq z_j + n_j.$$

This means that f_i, g_i and s_i are all zero by assumptions (8.41)-(8.43) and (8.44). This in turn means that the computed optimal point of the local optimization yields

$$x_i = 0 \qquad \text{for } z_j + 1 \leq i \leq z_j + n_j \tag{8.45}$$

since it is not locally optimal to waste lift gas by injecting gas with a small rate into wells, which don't produce with this rate of lift gas injection. The statement of the last sentence can be seen by looking at Figure 8.1. Assume that there $q_{g,1} > 0$ but that the gas lift allocation from the first gas manifold to wells is such that none of the wells produce. Then instead, it would be better to reduce $q_{g,1}$ to zero, keep the pressure p_4 unchanged, decrease q_3 by $q_{g,1}$ and reduce p_3 such that the pressure drop constraint between p_3 and p_4 is satisfied. If p_3 is smaller it means that the compressor has to work less, but then the compressor has to burn less gas. The less gas is burned, the more gas can be sold. *i.e.* q_2 can be increased by the amount of gas, which the compressor burns less, plus by $q_{g,1}$. This follows from (8.16), which is the mass balance constraint at the compressor. (Remember hat q_4 stands for the amount of gas burned by the compressor.) This change would increase the objective and all constraints would still be satisfied. All the arguments of this paragraph imply that if in Algorithm 8.4.1 y_t is feasible for some t and its x-component satisfies (8.45) for some j then all y_k for $k \geq t$ will also have x-component satisfying (8.45) for that j. This, of course, is a bad feature with a serious impact on the quality of solutions of Algorithm 8.4.1.

The following definition of $L(\lambda, \mu)$ is better than (8.29) for the dual solution approach of subproblem (8.24)-(8.28). This definition replaces $q_{g,j}$ in (8.27) by $\hat{q}_{g,j}$ and relaxes this equality constraint. (Recall the material of section 8.5)

$$
\begin{aligned}
L(\lambda,\mu) := \max_{\{x_i\}_{i=1}^n} \Bigg\{ & \sum_{i=1}^{n} f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \\
& - \sum_{j=1}^{k} \lambda_j \Big(\sum_{i \in Q(j)} s_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - S_j \Big) \\
& - \sum_{j=1}^{k} \mu_j \Big(\sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - q_{f,j} \Big) \\
& \qquad - \sum_{j=1}^{m} \mu_{k+j} \Big(\sum_{i=z_j+1}^{z_j+n_j} x_i \quad - \hat{q}_{g,j} \Big) \\
& : \sum_{i=z_j+1}^{z_j+n_j} x_i \leq q_j^* \quad \forall j, \quad x_i \geq 0 \quad \forall i \Bigg\}
\end{aligned}
\tag{8.46}
$$

In (8.29) $\lambda \in I\!R^k$, $\mu \in I\!R^k$ whereas in (8.46) $\lambda \in I\!R^k$, $\mu \in I\!R^{k+m}$. (8.46) looks like (8.29) except that the last line in (8.29) is changed and one more line is added. $\hat{q}_{g,j}$ is chosen such that it satisfies:

$$
\begin{aligned}
&\text{if } q_{g,j} > 0 \quad \text{then} \quad \hat{q}_{g,j} = q_{g,j}, \\
&\text{if } q_{g,j} = 0 \quad \text{then} \quad \hat{q}_{g,j} > 0
\end{aligned}
\tag{8.47}
$$

$$\text{and} \quad \hat{q}_{g,j} < x_i^*(p_{g,j}, p_{f,\pi(i)}) \quad \text{for all } i \text{ with}$$
$$z_j + 1 \leq i \leq z_j + n_j \quad \text{and } x_i^*(p_{g,j}, p_{f,\pi(i)}) > 0.$$

The purpose of replacing $q_{g,j}$ by $\hat{q}_{g,j}$ and relaxing the constraints $\sum_{i=z_j+1}^{z_j+n_j} x_i - \hat{q}_{g,j} = 0$ in $L(\lambda, \mu)$ is to avoid the kind of scenario described in the previous paragraph, where Algorithm 8.4.1 gets stuck with the zero allocation of lift gas to certain groups of wells. The non-relaxed constraints

$$\begin{aligned} \sum_{i=z_j+1}^{z_j+n_j} x_i &\leq q_j^* \quad \forall j, \\ x_i &\geq 0 \quad \forall i \end{aligned} \tag{8.48}$$

play the role of X in (7.5), *i.e.* these constraints define a set X over which the maximization in $\{x_i\}_{i=1}^n$ in (8.46) takes place. An essential condition for the dual method to work is that q_j^* is strictly greater than $\hat{q}_{g,j}$. The larger q_j^* is, the larger can be the constraint violation in (8.27) by $\hat{x}$ in Algorithm 8.4.1 step 4. If for a $\hat{x} \in E$ in Algorithm 8.4.1 step 4

$$\sum_{i=z_j+1}^{z_j+n_j} \hat{x}_i >> q_{g,j} \quad \text{for some } j \tag{8.49}$$

then this can be a problem for the local optimization in step 5. In this case it can well be, looking at Figure 8.1, that q_3 has to be increased by the local optimization to satisfy the mass balance constraint at node p_4. Other constraints are likely to imply a higher value for the pressures p_3, p_2 and a higher value for the gas flow q_1. It is not hard to see that if (8.49) is the case for a $\hat{x} \in E$ then the local optimization in step 5, taking $\hat{x}$ as a starting point, is likely to have difficulties in converging to a feasible point of problem (8.10)-(8.22). The other case that for some $\hat{x} \in E$ in step 4 of Algorithm 8.4.1

$$\sum_{i=z_j+1}^{z_j+n_j} \hat{x}_i << q_{g,j} \quad \text{for some } j$$

does not cause problems for the local optimization in step 5. In this case, looking at Figure 8.1, the local optimization can achieve feasibility by decreasing q_3, p_3 and increasing q_2, which even increases the objective function. The purpose of the non-relaxed constraint (8.48) in (8.46) is to limit the constraint violation of the kind (8.49). Some computational evidence for this argument will be given in the computational results section 8.7.1. Good experience has been made with choosing q_j^* in the following way:

$$\begin{aligned} A &= \max_i \{x_i^*(p_{g,j}, p_{f,\pi(i)}) \quad : \quad z_j + 1 \leq i \leq z_j + n_j\} \\ q_j^* &= \max\{A, \theta \hat{q}_{g,j}\} \quad \text{where } 1.2 \leq \theta \leq 2. \end{aligned} \tag{8.50}$$

In words: q_j^* should be at least as large as every discontinuity point x_i^* of the wells, which are connected to gas manifold j, and at least between $1.2\hat{q}_{g,j}$ and $2\hat{q}_{g,j}$.

It must be shown that $L(\lambda,\mu)$ as defined in (8.46) can be evaluated by DP. With definition (8.46) manipulations can be done, which are very similar to (8.32)-(8.36). Note that (8.30), (8.31) still hold unchanged. Let's define $T_j(\lambda,\mu)$. The definition below is similar to (8.33).

$$T_j(\lambda,\mu) := \max_{\{x_i\}_{i=z_j+1}^{z_j+n_j}} \left\{ \sum_{i=z_j+1}^{z_j+n_j} f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \right.$$
$$- \sum_{i=z_j+1}^{z_j+n_j} \lambda_{\pi(i)} s_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)})$$
$$- \sum_{i=z_j+1}^{z_j+n_j} \mu_{\pi(i)} g_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)})$$
$$-\mu_{k+j}\Big(\sum_{i=z_j+1}^{z_j+n_j} x_i \quad - q_{g,j}\Big)$$
$$\left. : \sum_{i=z_j+1}^{z_j+n_j} x_i \le q_j^*, \quad x_i \ge 0 \quad \forall z_j+1 \le i \le z_j+n_j \right\}$$

$L(\lambda,\mu)$ and $T_j(\lambda,\mu)$ are related, again, by (8.36). Therefore, if $T_j(\lambda,\mu)$ can be computed with DP then so can be $L(\lambda,\mu)$. Defining $\hat{f}_i(x_i)$ by (8.34)

$$T_j(\lambda,\mu) = \max_{\{x_i\}_{i=z_j+1}^{z_j+n_j}} \left\{ \sum_{i=z_j+1}^{z_j+n_j} \hat{f}_i(x_i) \quad - \mu_{k+j}\Big(\sum_{i=z_j+1}^{z_j+n_j} x_i \quad - \hat{q}_{g,j}\Big) \right.$$
$$\left. : \sum_{i=z_j+1}^{z_j+n_j} x_i \le q_j^*, \quad x_i \ge 0 \quad \forall z_j+1 \le i \le z_j+n_j \right\}.$$

In order to show that $T_j(\lambda,\mu)$ can be evaluated with DP let's define the value function

$$F_t(q) := \max_{\{x_i\}_{i=z_j+1}^{z_j+t}} \left\{ \sum_{i=z_j+1}^{z_j+t} \hat{f}_i(x_i) \quad : \right.$$
$$\left. \sum_{i=z_j+1}^{z_j+t} x_i = q \text{ and } x_i \ge 0 \quad \forall i \text{ with } z_j+1 \le i \le z_j+t \right\}$$

$F_t(q)$ has the DP recurrence relation:

$$F_t(q) = \max_{0 \le x_{z_j+t} \le q} \{\hat{f}_{z_j+t}(x_{z_j+t}) + F_{t-1}(q - x_{z_j+t})\}$$

Therefore $F_{n_j}(q)$ can be calculated for all $q \geq 0$ using DP. Notice that

$$T_j(\lambda, \mu) = \max_{0 \leq q \leq q_j^*} \{F_{n_j}(q) \quad - \mu_{k+j}(q - \hat{q}_{g,j})\}.$$

This shows that $T_j(\lambda, \mu)$ can be calculated with DP. Hence $L(\lambda, \mu)$ can be calculated using DP. The function $\hat{f}_i$ is at every point continuous and differentiable except at $x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)})$. Therefore FJ-DP can be applied for the computation of $F_{n_j}(q)$. As for the discretisation, a variable grid method is good and has been used.

Another problem of Algorithm 8.4.1 is similar to the problem described so far. This problem again is caused by the discontinuity points x_i^*. Let's look at

$$\sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) \tag{8.51}$$

for some j. This expression appears in (8.12) which describes the gas balance constraints at the flowstations. $p_{g,\phi(i)}$, $p_{f,j}$ and $q_{f,j}$ are variables in problem (8.10)-(8.22) but are fixed parameters in the subproblem (8.24)-(8.28). The set of values taken by (8.51), when $p_{g,\phi(i)}$, $p_{f,j}$ are fixed and x_i varies, is not connected in general. The reason for this is that $g_i(x_i, p_{g,\phi(i)}, p_{f,j})$, when considered as a function of x_i only, is not continuous everywhere and hence the set of values taken by (8.51), as x_i varies, is not connected in general. The reason for combining the dual method of subproblem (8.24)-(8.28) with a local optimization of problem (8.10)-(8.22) is to make possible changes in x_i across the discontinuity points x_i^*. Since $g_i \geq 0$ for all arguments the smallest value that (8.51) can have is zero.

Suppose that in Algorithm 8.4.1 after a local optimization $q_{f,j} = 0$ for some j and constraint (8.12) is satisfied by the present point y_t (for some t). This implies that

$$g_i(x_i, p_{g,\phi(i)}, p_{f,j}) = 0 \qquad \text{for all } i \in Q(j),$$

i.e. for all i corresponding to a well which is connected to flowstation j. When $g_i = 0$ it is very likely that

$$x_i < x_i^*(p_{g,\phi(i)}, p_{f,j}) \qquad \text{for } i \in Q(j),$$

which in turn implies $x_i = 0$ because it is not locally optimal to waste lift gas by injecting gas with a small rate into wells, which don't produce with this rate of lift gas injection. Then it is very likely that in the set of values, that

$$\sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - q_{f,j}$$

can take, zero is an isolated element and zero is the smallest element. But then it is better to replace $q_{f,j}$ in the definition of $L(\lambda,\mu)$ by $\hat{q}_{f,j}$ which satisfies:

$$\begin{aligned}
&\text{if } q_{f,j} > 0 \quad \text{then} \quad \hat{q}_{f,j} = q_{f,j}, \\
&\text{if } q_{f,j} = 0 \quad \text{then} \quad \hat{q}_{f,j} > 0 \\
&\qquad\qquad \text{and} \quad \hat{q}_{f,j} < g_i(x_i^*(p_{g,\phi(i)}, p_{f,j}), p_{g,\phi(i)}, p_{f,j}) \quad \text{for all } i \text{ with} \\
&\qquad\qquad\qquad\qquad \pi(i) = j \quad \text{and } g_i(x_i^*(p_{g,\phi(i)}, p_{f,j}), p_{g,\phi(i)}, p_{f,j}) > 0.
\end{aligned} \tag{8.52}$$

(Recall again section 8.5.) The best definition of $L(\lambda,\mu)$ is (8.46) with $q_{f,j}$ replaced by $\hat{q}_{f,j}$.

$$\begin{aligned}
L(\lambda,\mu) := \max_{\{x_i\}_{i=1}^n} \Bigg\{ & \sum_{i=1}^{n} f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \\
& - \sum_{j=1}^{k} \lambda_j \Big(\sum_{i \in Q(j)} s_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - S_j \Big) \\
& - \sum_{j=1}^{k} \mu_j \Big(\sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - \hat{q}_{f,j} \Big) \\
& - \sum_{j=1}^{m} \mu_{k+j} \Big(\sum_{i=z_j+1}^{z_j+n_j} x_i \quad - \hat{q}_{g,j} \Big) \\
& : \sum_{i=z_j+1}^{z_j+n_j} x_i \le q_j^* \quad \forall j, \quad x_i \ge 0 \quad \forall i \Bigg\}
\end{aligned} \tag{8.53}$$

Again, $\hat{q}_{f,j}$ is chosen such that it satisfies (8.52), $\hat{q}_{g,j}$ such that (8.47) and q_j^* such that (8.50). The dual method of solving subproblem (8.24)-(8.28) followed by the perturbation method (Procedure 7.3.1) works best when defining $L(\lambda,\mu)$ by (8.53). $L(\lambda,\mu)$ as defined in (8.53) can be evaluated by DP because (8.53) differs from (8.46) only in that $q_{f,j}$ is replaced by $\hat{q}_{f,j}$ and because $L(\lambda,\mu)$ as defined in (8.46) can be evaluated by DP (this has been shown before). For $L(\lambda,\mu)$ as defined in (8.53) there is, again, a subtangent available, which makes it possible to solve the dual problem (8.37)-(8.38) with a cutting plane method. This will be shown by the next theorem which is similar to Theorem 8.3.1.

Theorem 8.6.1 *Let $(\hat{\lambda},\hat{\mu}) \in \mathbb{R}^k \times \mathbb{R}^{k+m}$, let $\hat{x}$ be an argument which achieves the maximum in the definition of $L(\hat{\lambda},\hat{\mu})$ according to (8.53), i.e. $\hat{x}$ maximizes the RHS of (8.53) with λ,μ being replaced by $\hat{\lambda},\hat{\mu}$ respectively. Also let p be a $2k+m$-dimensional vector with components*

$$p_j \;=\; S_j - \sum_{i \in Q(j)} s_i(\hat{x}_i, p_{g,\phi(i)}, p_{f,j}) \quad \text{for } 1 \le j \le k,$$

$$p_{k+j} = \hat{q}_{f,j} - \sum_{i \in Q(j)} g_i(\hat{x}_i, p_{g,\phi(i)}, p_{f,j}) \quad \text{for } 1 \leq j \leq k,$$

$$p_{2k+j} = \hat{q}_{g,j} - \sum_{i=z_j+1}^{z_j+n_j} \hat{x}_i \quad \text{for } 1 \leq j \leq m.$$

Then p is a subtangent vector of the subdifferential set $\partial L(\hat{\lambda}, \hat{\mu})$, i.e.

$$L(\lambda, \mu) \geq L(\hat{\lambda}, \hat{\mu}) + p^T \begin{pmatrix} \lambda - \hat{\lambda} \\ \mu - \hat{\mu} \end{pmatrix}$$

for all $\lambda \in \mathbb{R}^k, \mu \in \mathbb{R}^{k+m}$.

Proof:
The proof is similar to the proof of Theorem 8.3.1. □.

8.7 Computational results

In this section some representative computational results of implementations of the Lagoven model are given. Two kinds of results are presented.

In the first subsection results are given of constructed problems, which basically means that the functions involved in problem (8.10)-(8.22) are constructed. However, these functions are constructed so as to have an analytical description and to have shapes and characteristics similar to the "real world" functions. The constructed problems allow to use a standard SQP subroutine as the local optimization routine in step 5 of Algorithm 8.4.1.

In the second subsection computational results of real world problems are presented. Edinburgh Petroleum Services Ltd. (EPS), an oil consultancy and software company, is interested in the solution of Lagoven-type problems. EPS has developed software which models pipes and oil wells. From given data of pipes, oil wells and compressors this software can piecewise linearly approximate pressure drop functions of pipes, production functions of wells and functions describing the compressor activity (like the function h in (8.17)). EPS also has a SLP local optimization code. Results of problems of the form (8.10)-(8.22) will be presented, where the functions f_i, s_i, g_i, $w_{f,i}$, w_i, h, w_2, $w_{g,i}$ are piecewise linear functions which correspond to real wells, pipes and compressors. EPS has developed an algorithm for problems of the type (8.10)-(8.22) which is a hybrid algorithm of Tabu Search and local optimization. The results of Algorithm 8.4.1 will be compared to those of the EPS hybrid algorithm.

8.7.1 Results of constructed problems

The results of this section are results of Algorithm 8.4.1 applied to the constructed problems P-Art1 and P-Art2, which are problems of the form (8.10)-(8.22). The functions f_i, s_i, g_i, $w_{f,i}$, w_i, h, w_2, $w_{g,i}$ and water capacities S_i of P-Art1 and of P-Art2 are given in the Appendix, section B. In this version of Algorithm 8.4.1 the initial point y_0 in step 2 is obtained by the method described in section 8.4.1, *i.e.* y_0 is the computed solution point of a smoothed problem. The optimal objective function value of the smoothed problem, let it be called $\tilde{A}$, is useful as a measure of the performance of Algorithm 8.4.1. Algorithm 8.4.1 tries to overcome the difficulties that a local optimization has with the discontinuous functions f_i, s_i, g_i. The smoothed problem replaces f_i, s_i, g_i by continuous functions $\tilde{f}_i$, $\tilde{s}_i$, $\tilde{g}_i$ which approximate f_i, s_i, g_i respectively. Therefore it can be expected that $\tilde{A}$, the computed optimal objective value of the smoothed problem, approximately is an upper bound on the optimal objective value computed by Algorithm 8.4.1. Table 8.1 gives the results of problem P-Art1 when step 1 of Algorithm 8.4.1 chooses $N = 10$. y_0 of Table 8.1 is feasible for the smoothed problem but infeasible for the actual problem. The objective value given for y_0 is the one corresponding to the smoothed problem. The definition of $L(\lambda, \mu)$ used corresponds to (8.53). Table 8.2 shows the results corresponding to problem P-Art2.

In Table 8.1 no convergence can be observed but in Table 8.2 it can be. Convergence is not expected in general. In Table 8.2 the algorithm only computed y_0, y_1, y_2, y_3 and y_4 because it found

$$||y_3 - y_4|| < \delta,$$

$\delta > 0$ being a small tolerance parameter, and hence the stopping condition in step 6 of Algorithm 8.4.1 led to termination.

An important observation is that some y_k $(1 \leq k \leq 10)$ have an objective function value that comes very close to or even surpasses the computed optimal objective function value of the smoothed problem. A second important observation is that in both tables (Table 8.1 and Table 8.2) the objective value at y_1 comes rather close to the computed optimal objective value of the smoothed problem, *i.e.* to the objective value given in the tables for $i = 0$. This is an observation which has been made in various other problems, some of these are shown in Table 8.3, and which suggests that the computed optimal solution point of the smoothed problem is a good starting point for Algorithm 8.4.1. In fact, due to this observation the results of subsection 8.7.2 are obtained using $N = 1$ as the parameter in step 1 of Algorithm 8.4.1.

Next, some computational evidence is given that it is good to include the non-

i	objective value at y_i
0	211.97
1	208.67
2	188.27
3	205.07
4	176.62
5	178.67
6	160.12
7	183.74
8	185.48
9	179.69
10	180.94

Table 8.1: Results of problem P-Art1, $L(\lambda, \mu)$ defined by (8.53)

i	objective value at y_i
0	263.28
1	261.73
2	155.23
3	264.0025
4	264.0029
5	(264.0029)
6	(264.0029)
7	(264.0029)
8	(264.0029)
9	(264.0029)
10	(264.0029)

Table 8.2: Results of problem P-Art2, $L(\lambda, \mu)$ defined by (8.53)

Problem name	number of wells, gas manifolds, flowstations	computed objective of smoothed problem	objective at y_1
Q1	25, 3, 3	274.23	264.01
Q2	25, 3, 3	283.78	281.38
Q3	30, 3, 3	289.41	282.27
Q4	30, 3, 3	326.14	318.33
Q5	30, 3, 4	399.27	376.76
Q6	30, 3, 4	418.44	411.44

Table 8.3: objective at y_1 is close to the computed optimal objective of the smoothed problem

relaxed constraints (8.48) in the definition of $L(\lambda, \mu)$ in (8.53). In section 8.6 this point has been discussed and arguments for the inclusion of constraints (8.48) in the definition of $L(\lambda, \mu)$ have been given. The following definition of $L(\lambda, \mu)$ is like (8.53) but without constraints (8.48).

$$
\begin{aligned}
L(\lambda,\mu) := \max_{\{x_i\}_{i=1}^n} \Bigg\{ & \sum_{i=1}^{n} f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) \\
& - \sum_{j=1}^{k} \lambda_j \Big(\sum_{i \in Q(j)} s_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - S_j \Big) \\
& - \sum_{j=1}^{k} \mu_j \Big(\sum_{i \in Q(j)} g_i(x_i, p_{g,\phi(i)}, p_{f,j}) \quad - \hat{q}_{f,j} \Big) \\
& - \sum_{j=1}^{m} \mu_{k+j} \Big(\sum_{i=z_j+1}^{z_j+n_j} x_i \quad - \hat{q}_{g,j} \Big) \quad : x_i \geq 0 \quad \forall i \Bigg\}
\end{aligned}
\tag{8.54}
$$

Table 8.4 shows the results when Algorithm 8.4.1 is used for problem P-Art1 with $L(\lambda, \mu)$ being defined by (8.54) instead of by (8.53), this being the only difference to the version which obtained the results of Table 8.1.

Table 8.5 shows the results when Algorithm 8.4.1 solves problem P-Art2 with $L(\lambda, \mu)$ being defined by (8.54) instead of by (8.53), this being the only difference to the version which obtained the results of Table 8.2.

The comparison of the results of Table 8.1 to those of Table 8.4 and of the results of Table 8.2 to those of Table 8.5 supports the arguments given for the inclusion of the non-relaxed constraints (8.48) in the definition of $L(\lambda, \mu)$. However, definition (8.54) does not give too bad results, either. The results of Table 8.4, for example, are only marginally worse than those of Table 8.1, and in Table 8.5 the maximum objective value of a y_k ($k \geq 1$) is 254.28 in comparison to 264.0029 in Table 8.2. However, notice that in Table 8.5 the objective value achieved by y_1 is only 209.62 in comparison to 261.73 in Table 8.2 which is unsatisfactory when expecting that the objective value at y_1 is close to the computed optimal objective value of the smoothed problem.

8.7.2 Results of EPS's test problems

In this section results of test problems from Edinburgh Petroleum Services Ltd are given. Algorithm 8.4.1 with $N = 1$ in step 1 is applied to six test problems, which are called P1, P2, ..., P6. These test problems are of the form (8.10)-(8.22), so that Figure 8.1 is a graphical description of these problems. The numbers of wells, gas manifolds and flowstations for each problem is given in Table 8.6. When Algorithm 8.4.1 evaluates $L(\lambda, \mu)$ (while solving the dual problem in step

i	objective value at y_i
0	211.97
1	208.59
2	163.30
3	165.39
4	205.98
5	164.79
6	165.39
7	205.99
8	164.79
9	165.39
10	206.44

Table 8.4: Results of problem P-Art1, $L(\lambda, \mu)$ defined by (8.54)

i	objective value at y_i
0	263.28
1	209.62
2	208.71
3	254.28
4	186.15
5	166.66
6	170.03
7	249.29
8	172.79
9	247.99
10	167.75

Table 8.5: Results of problem P-Art2, $L(\lambda, \mu)$ defined by (8.54)

4) by DP with a variable grid and FJ action elimination, the speed depends (about linearly) on the discretisation number. The test problems are run with discretisation numbers 30 and 100. The results also include runs of EPS's hybrid algorithm.

EPS's hybrid algorithm combines Tabu Search with local optimization. This algorithm can be briefly outlined as follows: The Tabu Search searches over the set of subsets of

$$\{1, 2, ..., n\},$$

i.e. it searches over the set of subsets of wells. (Recall that n is the number of wells in the problem.) When the Tabu Search chooses a subset I of $\{1, 2, ..., n\}$ it calls the local optimization routine which solves problem (8.10)-(8.22) with the additional constraints

$$\begin{aligned} x_i &\geq x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)}) \quad \forall i \in I \\ x_i &= 0 \quad \forall i \notin I. \end{aligned}$$

(Remember that the functions f_i, s_i, g_i are such that when $x_i < x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)})$ then it is at least as good to set $x_i = 0$.)

Table 8.7 shows the run times and the computed optimal objective value of EPS's Tabu Search and of Algorithm 8.4.1 with discretisation numbers 30 and 100 for the variable grid DP. Also the results of the local optimization of the smoothed problems are shown. The results suggest that, overall, the quality of solutions of Algorithm 8.4.1 is comparable with those of the Tabu Search. For large problems the run times are favourable for Algorithm 8.4.1, whereas for small problems the Tabu Search is faster.

When comparing the computed objective values two surprises can be observed. The first surprise is that in problem P4 Algorithm 8.4.1 with DP and discretisation number 100 yields an objective value which is about 45% larger than the result of the same method with discretisation number 30. This can be explained by the fact that when DP evaluates the Lagrangian $L(\lambda, \mu)$, even only a small change in precision (caused by the different discretisation number) can result in a corresponding optimal argument $x(\lambda, \mu)$ which is very different. In particular this different $x(\lambda, \mu)$ can represent a very different switch on-off pattern for the oil wells. This means that the next local optimization of the whole problem uses a different switch on-off pattern for the wells. This can be seen by looking at step 5 in Algorithm 8.4.1. A different optimal argument for $L(\lambda, \mu)$ in general means a different $\hat{x}$ in step a), this in turn means a different set I in step b) and hence a different local optimization in step c). It should also be noticed that in

problem	number of wells	number of gas manifolds	number of flowstations
P1	10	3	3
P2	13	4	3
P3	35	7	6
P4	50	7	6
P5	70	11	10
P6	100	18	16

Table 8.6: number of wells, gas manifolds, flowstations for each EPS test problem

problem	**local optimization on smoothed problem**		**Tabu Search**	
	run time	objective value	run time	objective value
P1	6.9(secs)	8.847520E-4	26(secs)	8.890263E-4
P2	24(secs)	6.701470E-4	36(secs)	7.024592E-4
P3	27(secs)	7.222792E-3	271(secs)	7.338544E-3
P4	48(secs)	9.962784E-3	1110(secs)	9.756659E-3
P5	29(secs)	1.014839E-2	3131(secs)	1.013346E-2
P6	95(secs)	1.238716E-2	10383(secs)	1.247742E-2

problem	**Algorithm 8.4.1 with DP discretisation number 30**		**Algorithm 8.4.1 with DP discretisation number 100**	
	run time	objective value	run time	objective value
P1	53(secs)	8.748920E-4	112(secs)	8.748920E-4
P2	115(secs)	7.027255E-4	242(secs)	7.027257E-4
P3	282(secs)	7.112659E-3	765(secs)	7.427997E-3
P4	666(secs)	9.912660E-3	1479(secs)	1.444160E-2
P5	1603(secs)	1.112866E-2	5357(secs)	1.101909E-2
P6	3199(secs)	2.032732E-2	6438(secs)	1.862933E-2

Table 8.7: Results of EPS's test problems

problem P4 Algorithm 8.4.1 with DP and discretisation number 100 yields an objective value which is about 43% larger than the results of Tabu Search and of the local optimization on the smoothed problem. Unlike in problem P4, in problem P6 Algorithm 8.4.1 with DP and discretisation number 100 yields an objective value which is worse than the result of the same method with discretisation number 30. The explanation for this is that when $L(\lambda, \mu)$ is calculated less accurately then Algorithm 8.4.1 can also return a different starting point for the local optimization, which in fact is a better starting point. There is an element of randomness. The important point is that the more accurately $L(\lambda, \mu)$ is calculated the larger the likelihood that Algorithm 8.4.1 finds a good starting point for a local optimization.

The second surprise in Table 8.7 are the results of problem P6. The computed optimal objective value of Algorithm 8.4.1 with DP and discretisation number 30 is more than 50% higher than the one computed by Tabu Search. This (and also the result of problem P4) is to be explained, I believe, by the search space becoming too large and too irregular ("many hills and valleys") for Tabu Search. The search space for Tabu Search in this problem has 2^{100} elements, one element being a subset I of the set $\{1, 2, ..., 100\}$. This search space is 2^{30} times larger than the search space of problem P5. One also has to remember that Tabu Search essentially is a random search doing local moves each time. In comparison, Algorithm 8.4.1 uses a lot more structure by using a dual method and by decomposing the problem. Also it is important to mention that the implementation of the Tabu Search looks at the computed optimal objective value of the smoothed problem as an indicator for when to stop. Table 8.7 shows this quite clearly. Hence, when there exists a solution of the problem with a much higher objective value than the computed optimal objective value of the smoothed problem then Tabu Search is likely to stop prematurely.

At least one of the versions of Algorithm 8.4.1 with DP always yields a higher objective value than Tabu Search does except for problem P1. The computation time of the Tabu Search increases faster with the size of the problem than Algorithm 8.4.1 with DP does.

When comparing the two versions of Algorithm 8.4.1 it can be observed that the run time of the version with discretisation number 100 is about 2-3.5 times the one of the version with discretisation number 30. Except for problems P3, P4 and P6 the computed optimal objective values of the two versions are similar. However, in problems P3 and in particular P4 the solution of the version with discretisation number 100 is a lot better, whereas in problem P6 it is worse.

8.8 Conclusions

In this chapter an oil production optimization problem has been discussed. In this problem some functions involved have discontinuities. To overcome the problems arising from the discontinuities Algorithm 8.4.1 has been proposed, which combines the dual method of a subproblem with local optimization.

It has been shown, by presenting results of several example problems, that Algorithm 8.4.1 gives good results. The Lagoven problem is a non-linear optimization problem with some functions involved having discontinuities and there are not many alternative suitable optimization methods. In particular, not many alternative methods have been implemented. Edinburgh Petroleum Services Ltd has developed a method combining Tabu Search and local optimization. Algorithm 8.4.1 has been compared to this alternative implementation. For large problems Algorithm 8.4.1 seems to give better results in shorter time whereas for small problems the Tabu Search appears to perform better.

Chapter 9

Conclusions and possible future work

The first half of the thesis presented an action elimination procedure for DP which uses first order optimality conditions and which reduces the run time by one order of magnitude for large problems. This action elimination technique was applied to the following classes of DP problems: discrete and continuous allocation problems, discrete and continuous stochastic problems with either finite or infinite horizon. Several examples with computational results were given.

First, the theory and basic principles of this action elimination procedure were introduced and explained for a class of deterministic and continuous allocation problems.

Then issues were discussed which are important for the efficient implementation of the action elimination procedure, like discretisation schemes and effective methods for finding solutions to the Fritz-John equations for any given Lagrange multipliers.

After that, deterministic discrete allocation problems were studied. For a class of discrete allocation problems with linear constraints necessary optimality conditions were derived. These optimality conditions are similar in form to the Fritz-John conditions for differentiable problems and were taken as the basis for action elimination.

Certain types of stochastic DP problems were discussed which allow action elimination based on first order optimality conditions. For problems with a reward function of a special form, the theory of the action elimination gave insight about the nature of the optimal solution by identifying target states. Water reservoir problems often have a reward function of this special form. This discussion, I feel, is far from being complete. Further classes of stochastic DP problems allowing action elimination based on first order optimality conditions can probably be found.

The second half of the thesis discussed a dual algorithm in a framework such that the results could be applied to an industrial problem. The main result of the discussion of the dual method was a perturbation procedure, which was added to the usual dual method. The usual dual method for solving a non-linear problem returns one point x of the primal search space. This point x need not be primal feasible but it can be a "good" point with respect to objective function value and constraint satisfaction and hence suited as a starting point for a local optimization. The perturbation procedure (Procedure 7.3.1) is a method to not only find one point x as the result of the dual method but several points, where each of these points is equally likely to be a good starting point for a local optimization. Therefore it is good to run, for each point x, a local optimization with x as starting point.

In chapter 8 an oil production optimization problem was discussed. This problem was decomposed and a subproblem was solved by the dual method with perturbation procedure. The algorithm developed was compared to another method which uses Tabu Search and local optimization. It was found that in general the algorithm developed performs better than the Tabu Search method, in particular for large problems. I believe that the dual method with the perturbation procedure added can be applied to other non-linear and discontinuous optimization problems. This could be a fertile area for future work. One such other possible application will be briefly outlined.

In the oil industry the following design problem arises, see Figure 9.1. There are a number of oil wells which naturally produce, *i.e.* no gas is injected into the wells. The wells can be connected to any one of the flowstations. The problem is to decide which well to connect to which flowstation. The wells produce oil and waste. Some of the waste is separated at the flowstations. When one connection pattern of wells-to-flowstations is fixed then the optimization is a continuous problem suitable for SLP or SQP. A natural subproblem is the optimization of the connection pattern when all other variables are fixed. Good connection patterns could possibly be found by the dual method with perturbation. Having decided on a new connection pattern a local optimization can be applied to the whole system. Then the continuous variables can be fixed again, the dual method with perturbation applied and hence the process can be repeated.

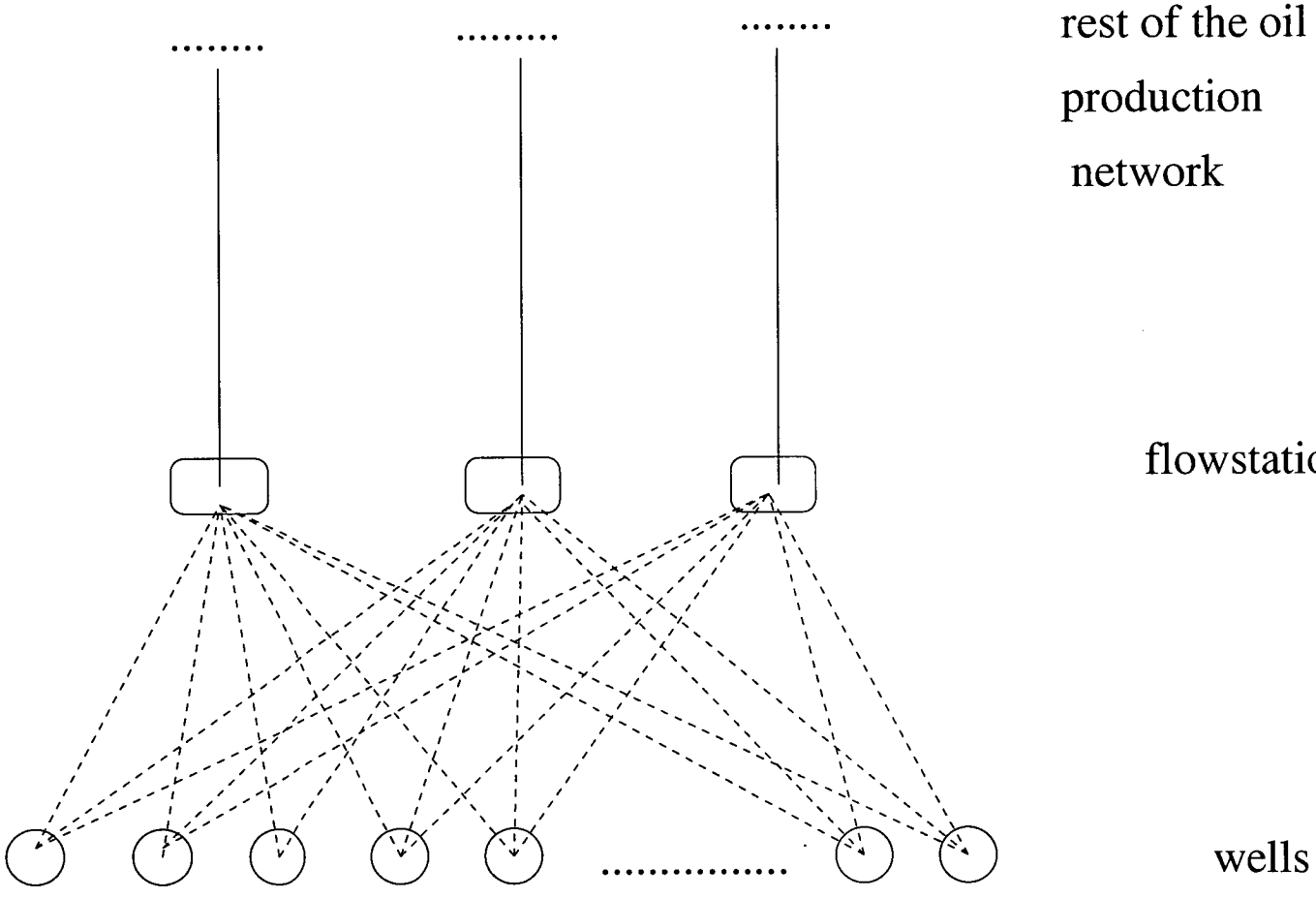

Figure 9.1: A connection problem

Appendix A

Algorithm A.0.1

1) Choose $n \in \mathbb{N}$ and set $\delta := \dfrac{\hat{q}}{n}$

$N_1 := \{x : x \in [0, d_1], f_1 \text{ or } w_1 \text{ is not differentiable at } x\}$

$S_1 := N_1 \cup \{0, d_1\}$

$Q := \left\{ \lfloor \frac{y}{\delta} \rfloor, \lceil \frac{y}{\delta} \rceil \quad : y \in S_1 \right\}$

for $k := 0$ to n set $F_1(k) := -\infty$

for $j := 0$ to n do

if $j\delta \leq d_1$ then

$k := \left\lceil \dfrac{w_1(j\delta)}{\delta} \right\rceil$

$\overline{F}_1(k) := f_1(j\delta)$

$\overline{A}_1(k) := j\delta$

$p_1(k) := w_1(j\delta)$

if $j \notin Q$ then

$\lambda(k, 1) := \dfrac{f_1'(j\delta)}{w_1'(j\delta)}$

else

$\lambda(k, 1) := undefined$

end if

end if

end do j

2) $i := 2$

3) for $k := 0$ to n set $\overline{F}_i(k) := -\infty$

$N_i := \{x : x \in [0, d_i], f_i \text{ or } w_i \text{ is not differentiable at } x\}$

$S_i := N_i \cup \{0, d_i\}$

$Q := \left\{ \lfloor \frac{y}{\delta} \rfloor, \lceil \frac{y}{\delta} \rceil \quad : y \in S_i \right\}$

for $j \in \{0, 1, ..., n\}$ for which $F_{i-1}(j) > -\infty$ do

if $\lambda(j, i-1)$ is defined then

$J := \left\{ x : x \in I\!R, 0 < x \leq d_i, f_i'(x) = \lambda(j, i-1) w_i'(x) \right\}$

for all $x \in J \cup S_i$ do

$k := \lceil \frac{p_{i-1}(j) + w_i(x)}{\delta} \rceil$

if $k \leq n$ and $\overline{F}_i(k) < \overline{F}_{i-1}(j) + f_i(x)$ then

$\overline{F}_i(k) := \overline{F}_{i-1}(j) + f_i(x)$

$\overline{A}_i(k) := x$

$p_i(k) := p_{i-1}(j) + w_i(x)$

$\lambda(k, i) := \lambda(j, i-1)$

end if

end do x

else (if $\lambda(j, i-1)$ is not defined)

for $t := 0$ to n do

$k := \lceil \frac{p_{i-1}(j) + w_i(t\delta)}{\delta} \rceil$

if $k \leq n$ and $\overline{F}_i(k) < \overline{F}_{i-1}(j) + f_i(t\delta)$ then

$\overline{F}_i(k) := \overline{F}_{i-1}(j) + f_i(t\delta)$

$\overline{A}_i(k) := t\delta$

$p_i(k) := p_{i-1}(j) + w_i(t\delta)$

if $t \notin Q$ then

$\lambda(k, i) := \frac{f_i'(t\delta)}{w_i'(t\delta)}$

else (if $t \in Q$)

$\lambda(k, i) := undefined$

end if

end if

end do t

end if

end do j

4) if $i < r$ increase i by one and return to 3)

5) if $F_r(n) > -\infty$ do

$s := n$

$\tilde{p} := p_r(n)$

Appendix B

The problems of subsection 8.7.1 use the following types of functions in (8.10)-(8.22): The pressure drop functions w_1 and w_2 (in (8.15), (8.18)) are of the form

$$\begin{aligned} w_1(p_1, q_1, p_2) &= p_1^2 - q_1^2 - p_2^2, \\ w_2(p_3, q_3, p_4) &= p_3^2 - q_3^2 - p_4^2. \end{aligned}$$

The pressure drop functions $w_{f,i}$ and $w_{g,i}$ (in (8.13), (8.20)) are of the form

$$\begin{aligned} w_{f,i}(p_{f,i}, q_{f,i}, p_1) &= p_{f,i}^2 - \frac{q_{f,i}^2}{i} - p_1^2, \\ w_{g,i}(p_4, q_{g,i}, p_{g,i}) &= p_4^2 - \frac{q_{g,i}^2}{i} - p_{g,i}^2. \end{aligned}$$

The function h describing the compressor activity (in (8.17)) is taken as

$$h(p_2, p_3, q_1, q_4) = (p_3 - p_2)q_1 - 500q_4(q_4 + 2).$$

The discontinuity point x_i^* and the well production functions f_i, s_i, g_i depend on their arguments and on three well specific parameters a_i, b_i, c_i.

$$x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)}) = 10 + \frac{50}{1 + e^{\frac{1}{20a_i}\left(p_{g,\phi(i)} - \frac{p_{f,\pi(i)}}{5}\right)}}$$

$$\tilde{x}_i(p_{g,\phi(i)}, p_{f,\pi(i)}) = 60 + \frac{a_i}{1 + e^{\left(\frac{p_{f,\pi(i)} - p_{g,\phi(i)}}{15a_i}\right)}}$$

$$f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) = \begin{cases} 0 & \text{if } x < x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)}) \\ 20 + (x_i + 1)^{b_i} + a_i\left(\frac{p_{g,\phi(i)}}{100} + \frac{1}{2}\right)^{c_i} & \text{if } x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)}) \le x_i \\ & \quad \le \tilde{x}_i(p_{g,\phi(i)}, p_{f,\pi(i)}) \\ f_i(\tilde{x}_i(p_{g,\phi(i)}, p_{f,\pi(i)}), p_{g,\phi(i)}, p_{f,\pi(i)}) & \text{if } \tilde{x}_i(p_{g,\phi(i)}, p_{f,\pi(i)}) < x_i \end{cases}$$

$$s_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) = \begin{cases} 0 & \text{if } x < x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)}) \\ (x_i+1)^{c_i} + a_i \left(\frac{p_{g,\phi(i)}}{100} + \frac{1}{2}\right)^{b_i} & \text{if } x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)}) \le x \\ \quad + f_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) & \le \tilde{x}_i(p_{g,\phi(i)}, p_{f,\pi(i)}) \\ s_i(\tilde{x}_i(p_{g,\phi(i)}, p_{f,\pi(i)}), p_{g,\phi(i)}, p_{f,\pi(i)}) & \text{if } \tilde{x}_i(p_{g,\phi(i)}, p_{f,\pi(i)}) < x \end{cases}$$

$$g_i(x_i, p_{g,\phi(i)}, p_{f,\pi(i)}) = \begin{cases} 0 & \text{if } x < x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)}) \\ x_i + (x_i+1)^{c_i} + a_i \left(\frac{p_{g,\phi(i)}}{100} + \frac{1}{2}\right)^{b_i} & \text{if } x_i^*(p_{g,\phi(i)}, p_{f,\pi(i)}) \le x \\ & \le \tilde{x}_i(p_{g,\phi(i)}, p_{f,\pi(i)}) \\ g_i(\tilde{x}_i(p_{g,\phi(i)}, p_{f,\pi(i)}), p_{g,\phi(i)}, p_{f,\pi(i)}) & \text{if } \tilde{x}_i(p_{g,\phi(i)}, p_{f,\pi(i)}) < x \end{cases}$$

The two artificial problem, called P-Art1 and P-Art2, have twenty wells, two gas manifolds and three flowstations. The parameters of wells a_i, b_i, c_i and connections of wells to gas manifolds and flowstations (*i.e.* $\phi(i)$ and $\pi(i)$) are given in Table B.1. The gas price in the objective function is

$$c = 0.6$$

The two problems P-Art1 and P-Art2 are different in that they have different water capacities at the flowstations. For their optimal solutions the water capacity constraints are active. Problem P-Art1 has water capacities

$$\begin{aligned} S_1 &= 50, \\ S_2 &= 90, \\ S_3 &= 90. \end{aligned}$$

Problem P-Art2 has water capacities

$$\begin{aligned} S_1 &= 80, \\ S_2 &= 90, \\ S_3 &= 120. \end{aligned}$$

i	a_i	b_i	c_i	$\phi(i)$	$\pi(i)$
1	20	0.62	0.35	1	2
2	10	0.7	0.7	1	1
3	15	0.6	0.3	1	3
4	10	0.45	0.3	1	1
5	20	0.3	0.2	1	2
6	15	0.5	0.4	1	3
7	10	0.5	0.3	1	2
8	7	0.5	0.4	1	1
9	10	0.65	0.4	1	3
10	8	0.8	0.7	1	1

i	a_i	b_i	c_i	$\phi(i)$	$\pi(i)$
11	15	0.5	0.8	2	2
12	5	0.85	0.6	2	3
13	20	0.5	0.5	2	1
14	15	0.4	0.7	2	2
15	10	0.6	0.5	2	3
16	7	1	0.5	2	2
17	10	0.5	0.4	2	1
18	8	0.8	1	2	3
19	15	0.45	0.5	2	3
20	5	0.8	0.9	2	1

Table B.1: Data for f_i, s_i, g_i in P-Art1 and P-Art2

Bibliography

[1] R. E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[2] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, 1987.

[3] E. V. Denardo. *Dynamic Programming Models & Applications.* Prentice-Hall, 1982.

[4] E. V. Denardo and B. L. Fox. Shortest-route methods: Reaching, pruning and buckets. *Operations Research*, 27:161–186, 1979.

[5] H. Everett. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, (11):399–413, 1963.

[6] E.W.Steiner and K.I.M.McKinnon. Dynamic programming using the Fritz-John conditions. To appear in *European Journal of Operational Research.*

[7] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, 1987.

[8] B. Fox and D. Landi. Searching for the multiplier in one-constrained optimization problems. *Operations Research*, 18:253–262, 1970.

[9] P. Gilmore and R. Gomory. The theory and computation of knapsack functions. *Operations Research*, 14:1045–1074, 1966.

[10] S. Mokhtar, D. Hanif, and C. Shetty. *Nonlinear Programming, Theory and Algorithms.* John Wiley & Sons, second edition, 1993.

[11] G. Nemhauser. *Introduction to Dynamic Programming.* Wiley, 1966.

[12] N. Nishikiori, R. Redner, D. Doty, and Z. Schmidt. An improved method for gas lift allocation optimization. *Journal of Energy Resources Technology-Transactions of the ASME*, 117(2):87–92, 1995.

[13] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 1994.